VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra 456

Zpracování modelu softwarových procesů ve firmě
Description of the Software Process Model

2011 **Bc. Michal Slíva**

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 15.8.2011 Podpis:

Souhlasím se zveřejněním této diplomové práce dle požadavků čl.26,odst. 9 studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 15.8.2011 Podpis:

Poděkování

Tímto bych chtěl poděkovat majiteli společnosti netdevelo s.r.o, Petrovi Svobodovi za to, že mi umožnil použít společnost netdevelo s.r.o. jako podklad pro tuto práci.

Abstrakt

Tato práce se věnuje popisu tradičních a agilních metod. Obzvláště se pak věnuje metodám Open UP, SCRUM a extermní programování. Popisuje rozdílu mezi těmito metodami a ukázky použití těchto metod na praktických příkladech.

V práci je popsán reálný model procesů ve firmě Netdevelo s.r.o. Procesy jsou upraveny za použití software Eclips Process Framework Composer. Tyto procesy jsou pak dále zpracovány a pomocí nabitých znalostí z oblasti Open UP, SCRUM a externího programování tak, aby byly vylepšeny oproti stávajícím.

V závěru zhodnotím nasazení těchto pozměněných procesů a ohodnotíme rozdíly mezi tradičními a agilními metodami vývoje softwaru.

Klíčová slova

Agilní, tradiční, metodika, procesní framework, Vodopádový model, OpenUP, Scrum, XP, extrémní programování, EPF, EPF Composer, proces, role, artefakt.

Abstract

This work is about to the description of agile methods, OpenUP, SCRUM and external programming. Description of the difference between these methods and examples of using these methods to practical examples.
The paper describes the model of real processes in the companynetdevelo Ltd. Processes are adjusted using the SoftwareProcess Framework Composer Eclips. These processes arethen further processed and charged with knowledge of the OpenUP, SCRUM and external programming so that they are improvedover existing.
In conclusion, we evaluate the deployment of these amendedprocedures are tested differences between traditional and agilesoftware development methods.

Key words

Agile, tradition, methodology, process framework, waterfall model, OpenUP, Scrum, XP, extreme programming, EPF, EPF Composer, process, role, artifact.

Slovník

AJAX	Asynchronous JavaScript and XML.
CSS	Cascading Style Sheets.
e-shop	Označení pro aplikaci (software) internetového obchodu.
HTML	HyperText Markup Language.
MVC	Model View Cotroller.
MySQL	Databázový systém.
RUP	Rational Unified Process.
Smarty	Šablonovací systém napsaný v PHP.
Stakeholder	označuje osobu, které se nějakým způsobem dotýká právě vyvíjený software. Obvykle označuje zákazníka.
NASA	National Aeronautics and Space Administration, Národní úřad pro letectví a kosmonautiku
XML	Extensible Markup Language.
PHP	Skriptovací programovací jazyk

OBSAH:

1. Úvod 8
1.1. Motivace 8
2. Historie vývoje software 8
2.1. Prvopočátky 8
Z1 9
Z2,Z3 9
Colosesus 9
ABC 9
Mark I 9
Mark II 9
ENIAC 10
Druhá generace 10
UNIVAC 10
Třetí generace (1965 až 1980) 10
Čtvrtá generace (od roku 1981) 10
2.2. Historie vývoje software 11
3. Metodiky vývoje software 12
3.1. Metodika 12
3.2. Procesní Framework 12
3.3. Úkolocentrické metody 12
3.3.1. Vodopádový model vývoje software 12
3.4. Hodnocentrické metody 14
4. OpenUp 16
4.1. Kde nasadit openUp? 17
4.2. Slovníček pojmů 18
4.3. Vrstvy OpenUp 20
4.5. Základní princip OpenUp 21
4.5.1. Spolupráce, jež je založena na porozumění a společných zájmech 21
4.5.2. Výběr/vývoj architektury 21
4.6. Proces 22
5. SCRUM 25
5.2. Artefakty 28

5.3. Schůzky ... 31
5.4. Proces v Scrum ... 32
6. Extremní programování ... 33
5.2. Základní informace ... 33
6.2. Role ... 36
6.3. Artefakty ... 37
6.4. Základní principy ... 38
6.4.1. Komunikace ... 38
6.4.3. Zpětná vazba ... 39
6.4.4. Odvaha ... 39
6.5. Praktiky XP ... 39
6.5.1. Programovací praktiky ... 39
6.6. Proces v XP ... 46
6.6.3. Příklady použití jednotlivých metodik ... 48
7. Postup podle OpenUp ... 49
7.2. Počáteční fáze ... 50
8. Postup vývoje dle SCRUM ... 52
9. Postup podle extremního programování ... 54
10. Rozdíly jednotlivých metodik ... 57
11. Eclipse Process Framework(EPF) ... 59
11.1. Základní informace o EPF ... 59
12. Popis procesů v netdevelo s.r.o. ... 60
12.3. Tvorba a úprava nových a stávajících projektů ... 62
13. Shrnutí aktuálního procesu ... 62
14. Návrhy na zlepšení aktuálních procesů ... 63
14.1. Kritické faktory procesu ... 63
14.2. Týdenní reporty ... 63
14.3. Knowledgebase ... 64
14.4. Schůzky realizačních oddělení ... 64
14.5. Refaktorování a testy ... 65
14.6. Standardy v kódu ... 65
15. Závěr ... 65
16. Seznam použité literatury ... 66
17. Přílohy ... 67

1. Úvod

V dnešní době se můžeme se softwarem setkat prakticky na každém kroku. Cesta k němu je ale mnohdy velmi trnitá. Většina programátorů začne jako samostatná jednotka. Jediný člověk stojí za zpracováním zadání, návrhem řešení, samotnou realizací a představením produktu, včetně jeho testování a konečným nasazením do provozu. Tento jediný člověk pak provádí často i následnou údržbu a provoz systému.

Později se takový člověk dostává do firemního prostředí, kde se často setkává s velmi odlišným směrem uvažování. Ve firemním prostředí se seznamujeme se zavedenými procesy vývoje software. Tyto procesy jsou plánovány a řízeny tak, abychom došli k co možná nejlepšímu cíli a přitom nedocházelo ke zmatkům a nejasnostem. Mezi typické problémy patří špatné pochopení zadání zákazníka. To způsobí předražení finálního výrobku, neefektivitu práce a zpoždění celého konečného produktu.

Dnes již máme za sebou několik etap efektivního vývoje software. Z těchto etap vznikl bezpočet metod, frameworku a sad best practices; jak vyvíjet software efektivně.

1.1. Motivace

Tuto práci jsem začal psát na základě toho, že již 4 roky pracuji v softwarové firmě. Ta v době mého nástupu patřila mezi velice malé. Dnes má kolem 40 zaměstnanců. V této společnosti šlo přímo ukázkově vidět, jak vyvstala potřeba zavedení procesů.

Ze začátku byla ve firmě dostačující ústní komunikace a vysvětlení, jak se dobrat zdárného konce. Dnes, z důvodu kapacit místnosti, již nemůžou všichni zaměstnanci sedět v jedné místnosti a tudíž není možné si takové informace ústně předávat. Firma se začala dělit na jednotlivá oddělení a nastala opravdová nutnost zavést procesy. Jelikož bez těchto procesů by se firma stávala neefektivní a vedoucí by neměli přehled nad tím, co jednotliví zaměstnanci dělají.

2. Historie vývoje software

2.1. Prvopočátky

První počítač byl vyroben v 30. letech 20. století, avšak za jeho vynálezce je přesto považován Charles Babbage, který již v 19. století vymyslel základní principy fungování stroje pro řešení složitých výpočtů. Tyto počítače z 19. století byly založeny na mechanickém principu. Ale toto nejsou počítače v pravém slova smyslu tak jak je chápeme v dnešní době.

Námi známé počítače jsou počítače tzv. nulté generace a jsou to přístroje založené na elektromechanické s využitím relé. Kmitočet, kterého byly schopny dosáhnout bylo okolo 100Hz. To vystačovalo až do druhé světové války. V době války vyvstala potřeba mnohem výkonnějších strojů a to z armádních důvodů.

Z1

První počítač německé výroby. Zprovozněn v roce 1936. Sestrojen Konradem Zuse. Počítač pracoval v dvojkové soustavě s aritmetikou s plovoucí desetinou čárkou. Program se vkládal na děrném štítku (kinofilm). Neobsahoval však podmíněné skoky.

Z2,Z3

Stejný autor jako Z1, obsahoval již 200 relé. Paměť nadále mechanická. Z3 spolupráce na vývoji s Helmutem Schreyrem. Dokončen 1941. Již 2600 relé a byl první prakticky použitelný počítač. Z3 se používal k výpočtu trajektorie balistických raket V2.

Colosesus

Anglická výroba rok 1943, sloužil k prolamování německých šifer šifrovacího stroje Enigma. Počítač používal vakuové elektronky.

ABC

Sestrojen americkým fyzikem k počítání lineárních rovnic.

Mark I

Financován IBM v provozu 1939-1944. Počítač pracoval na Harvardu a sloužil k demonstraci technických možností. Mark I byl 15m dlouhý a 3,7KW výkonným motorem poháněný. Obsahoval 767 000 různých elektromechanických prvků. Data četl z děrné pásky. Obsahoval statickou a dynamickou paměť. I když byl počítač na univerzitní půdě používalo jej americké námořnictvo pro výpočet drah balistických raket.

Mark II

Po úspěchu, který sklidil MARK I, sestrojili počítač sestavený pouze čistě pomocí relé. Aritmetika pracovala v plovoucí čárce s desítkovými číslicemi, které byly dvojkově kódovány pomocí čtyř relé. Operační paměť počítače mohla pojmout až 100 čísel s deseti platnými číslicemi. Sčítání již trvalo pouze 0,125 s a násobení průměrně 0,25 s. Celý počítač obsahoval

přibližně 13 000 relé. Počítač začal pracovat v roce 1947 a byl předán americkému námořnictvu.

ENIAC

Patří do tzv. první generace počítačů. Ta používala hlavně elektronky a v menší míře i relé. Počítače byly poměrně neefektivní, velmi drahé, měly vysoký příkon, velkou poruchovost a velmi nízkou výpočetní rychlost. Zpočátku byl program vytvářen na propojovacích deskách, později byly využity děrné štítky a děrné pásky, které též sloužily spolu s řádkovými tiskárnami k uchování výsledků. ENIAC pracoval na univerzitě v Pelsynvanii od roku 1944. Byl to první počítač, který pracoval podobně jako dnešní počítače. (turingově kompletní)

Druhá generace

Počítače druhé generace charakterizuje použití tranzistorů.

UNIVAC

UNIVAC byl v roce 1951 prvním sériově vyráběným komerčním počítačem a byl zkonstruován tvůrci počítače ENIAC.

Třetí generace (1965 až 1980)

Třetí generace je charakteristická použitím integrovaných obvodů. S postupem času roste počet tranzistorů v integrovaném obvodu (zvyšuje se integrace). V této době byl výkon počítače úměrný druhé mocnině jeho ceny, takže se vyplatilo koupit co nejvýkonnější počítač a poté prodávat jeho strojový čas. Majitelé požadovali maximalizaci využití počítače, proto se objevilo multiprogramování – zatímco jeden program čeká na dokončení I/O operace, je procesorem zpracovávána druhá úloha. S tím úzce souvisí zavedení pojmu proces, který označuje prováděný program a zahrnuje kromě něj i dynamicky se měnící data. Objevuje se první podpora multitaskingu, kdy se programy vykonávané procesorem střídají, takže jsou zdánlivě zpracovávány najednou. Tento pokrok umožňuje zavedení interaktivních systémů (počítač v reálném čase reaguje na požadavky uživatele).

Čtvrtá generace (od roku 1981)

Čtvrtá generace je charakteristická mikroprocesory a osobními počítači. Mikroprocesory zvýšily spolehlivost, zmenšily rozměry, zvýšila se rychlost a kapacita pamětí. Nastává ústup střediskových počítačů ve prospěch pracovních stanic a v roce 1981 uvedení osobního

počítače IBM PC. Přichází éra systémů DOS a vznikají grafická uživatelská rozhraní. Poměr cena/výkon je nejlepší u nejvíce prodávaných počítačů, vyšší výkon je vykoupen exponenciálním růstem ceny, proto se již nevyplatí koupit nejvýkonnější počítač na trhu a z mnoha běžných a laciných počítačů vznikají clustery. S rozvojem počítačových sítí vzniká Internet, distribuované systémy.

2.2. Historie vývoje software

Nyní jsme si načrtli, jak vypadala historie vývoje počítačů po hardwarové stránce. S Hardwarem je velice úzce spjat samozřejmě i software. Ve chvíli, kdy se počítače rozšířily i do firem a státních institucí, vyvstal nový problém. A to je problém mnohem složitějších úkonů, které tyto subjekty požadovaly. V tuto chvíli vznikla otázka vývoje softwaru, protože se musel začít dělat na míru 3. straně. Objevil se problém efektivity a tomu všemu se říká *problém softwarového inženýrství*.

„Softwarové inženýrství je činnost zahrnující inženýrství, informatiku a management, jejím cílem je návrh, tvorba a údržba počítačových programů.“ Software a jeho vývoj spolu se softwarovým inženýrstvím si prošli mnohými vývojovými etapami. Ty se odehrály v posledních 50 letech.

1. *Do roku 1956* - Programy se vyvíjely jen pro jeden konkrétní stroj. Tyto programy byly fyzicky vloženy do počítače pomocí *Určitého zapojení.* Takže byly neměnné a po naprogramování již neprocházely žádnou další etapou vývoje.
2. *Do roku 1970* - Položen základní pilíř softwarového inženýrství. Byl popsán a aplikován vodopádový model vývoje software.
3. *Do roku 1980* - Zavedeny postupy jako je návrh softwaru, specifikace, testování, následná údržba atd. Tyto postupy vznikly z důvodu potřeby trhu, kdy se začaly objevovat problémy nekonečných projektů. V této době se objevil i nový druh prodeje software. Začal se prodávat jak celý software, tak jen jeho části, kterým říkáme knihovny.
4. *Do roku 1990* - Vývoj softwaru a softwarové inženýrství zažívá obrovský rozkvět. Rozvíjejí se metodiky vývoje software, standardizace objevují se první ucelené architektury, frameworky. Software se již nepíše pouze na jednorázové použití.

3. Metodiky vývoje software

3.1. Metodika

Obecnější termín, než procesní Framework. Metodiky jsou podmnožinou Frameworků. Je to návod sada doporučených praktit a postupů, jak postupovat při vývoji software. Metodika pokrývá celý životní cyklus vytváření software.

3.2. Procesní Framework

První Framework byl představen v roce 1960. Pod jménem SDLC. SDLC mělo za úkol nastavit vývoj informačního systému tak aby byl účelný, strukturovaný a metodický. To znamená, že všechny fáze návrhu informačního systému od jeho návrhu až po údržbu byly provedeny postupně, podle přesně daného schématu. Tyto fáze se provádí vždy v nezměněné formě. Frameworky slouží k budování velice rozsáhlých systému.

Některé ze frameworků: IBM – RUP, AUP (Agile Unified Process), extremní programování, Scrum.

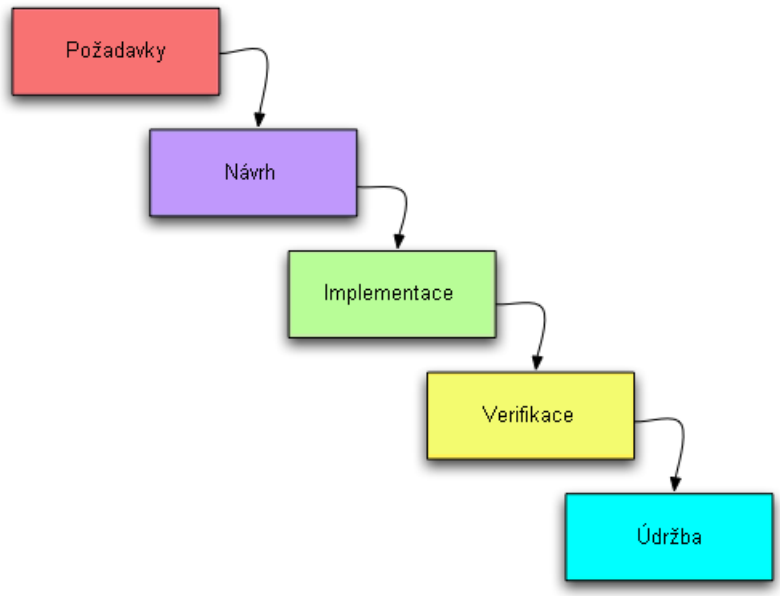
V dnešní době je dostupné velké množství frameworku. Každý z nich je vhodný k řešení jiného problému. Vhodný tip frameworku vybíráme podle druhu projektu, použité technologie, organizace apod.

3.3. Úkolocentrické metody

Za úkolocentrické metody se považují takzvané tradiční metody vývoje software. U této metody jde o přesné splnění zadaných úkolů a tím dosáhnutí stanoveného cíle. Úkoly zadává pověřena osoba a ta určuje jejich prioritu, datum ukončení napojení na ostatní úkoly atd. Projekt se představuje zadavateli až v jeho finální podobě. Je nutný velice silný důraz na naprosto přesnou specifikaci softwarového díla.

3.3.1. Vodopádový model vývoje software

Vodopádový model sekvenční proces vývoje software. A jak jeho název napovídá, jde o metodu, kdy je vývoj řešen sekvenčně. Tudíž krok za krokem. Vodopádový model byl popsán Winstonem W. Roycem v roce 1970 a to jako chybná metoda vývoje software, která je jako model nefunkční. Vodopádový model se skládá z několika kroků:



Obrázek 1: Vodopádový model

V principu se tedy přechází od jednoho kroku k druhému. Přičemž k dalšímu kroku se přechází až po naprosto striktním splnění kroku předcházejícího. Kupříkladu krok Návrhu aplikace můžeme provést až je naprosto přesná specifikace požadavků. Jsou tedy sepsány veškeré požadavky klienta a posléze předány k Návrhu.

3.3.2. Klady

Z praxe je známo, že opravení chyby je v počáteční fázi mnohem levnější než ve fázi pozdější. A to i několiksetkrát násobně. Vodopádový model byl měl zajistit, že v počáteční fází bude specifikace a návrh a implementace natolik přesná, že tyto chyby do jisté míry budou eliminovány. Vodopádový model by měl zajistit, že ve fázi návrhu se určí možnosti implementace a použitých technologií a v pozdější fázi nedojde k tomu, že se zjistí, že některé části jsou nekompatibilní a tudíž celá dlouhá práce přijde vniveč. Avšak u tohoto modelu se může stát, že samotný návrh nebo specifikace, z níž návrh vychází je špatný, a tudíž všechna následující práce může přijít na zmar.

Vodopádový model má také výhodu pro nově příchozího člena týmu v tom, že si přečte specifikaci, případně návrh, a je kompletně obeznámen s problémem.

Vodopádový model se oceňuje také pro svou jednoduchost a disciplínu. Postupuje čistě lineárně. Jednotlivé fáze jsou velice jasné a přímočaré. U projektu je velmi snadné si stanovit jednotlivé milníky.

Vodopádový model je vhodné používat tam, kde je zaručena neměnnost výsledného produktu a oplatí se vynaložit nemalé prostředky v prvních dvou fázích. Na oplátku bude výsledný produkt skutečně odpovídat počáteční specifikaci.

3.3.3. Zápory

Tento prototyp má velký zástup kritiků a sám autor jej představil jako chybný. Nejvíce se mu vyčítá, že je v praxi nepoužitelný. A to z důvodu, že nejsme schopni v počátečních fázích komplexně popsat projekt a nebo zadavateli nemusí být jasné veškeré aspekty. Zadavatel si většinou totiž uvědomí hodně drobností až po předvedení. A pak je velice nákladné projekt předělat a počáteční, velmi nákladná, část návrhu a specifikace se stává zbytečnou a nepoužitelnou.

Další problém přichází ve chvíli, kdy si návrhář neuvědomí možné problémy při implementaci. Zvláště u typů projektů, které se tvoří poprvé.

Jelikož výše popsaný modul častokrát nevyhovoval, doznal mnoha modifikacím. Asi nejznámější modifikaci se představíme.

3.3.4. Sašimi model

Autorem je Peter DeGracem. A krom Sašimi model se mu také říká vodopádový model s překrývajícími se fázemi, nebo vodopádový model se zpětnou vazbou. V principu jde pouze o to, že jednotlivé fáze se překrývají a tím je zmírněna míra následků při selhání některé z části. Například u Návrhu a implementace se pracuje částečně souběžně a tudíž požadavky, které se objeví při implementaci, se stačí zanést do návrhu.

3.4. Hodnocentrické metody

Jsou to tzv. agilní metody vývoje software. Je to iterační způsob vývoje software. Kdy se software vyvíjí v jednotlivých krocích (iteracích). Jednotlivé iterace se presentují zákazníkovi. Ten vidí, jak se jim zadaný projekt vyvíjí, jelikož v každé iteraci je produkt funkční. V každém kroku se přidává určitá hodnota(funkčnost software). Je zde relativní prostor na včasné připomínky od zadavatele.

V roce 2001 sepsalo 17 programátorů z různých části světa dokument; manifest agilního vývoje software. Úmyslem tohoto dokumentu bylo sepsat cíle jak vyvíjet software kvalitněji, rychleji a efektivněji potažmo levněji.

1. Bude dodán správný software
2. Software bude správně udělán
3. Software bude vyroben rychle
4. Software bude vyroben hospodárně

Existuje mnoho přístupů, jak těchto cílů dosáhnout. Některé se striktně drží uvedených kroků. Podstatné ale je uvědomit si, že je třeba se soustředit na důležité milníky, které se presentují klientovi namísto presentování věci nepodstatných a tím mrhat čas vývojáře. Protože čas strávený vývojářem na projektu je nejdražší položka. Procesy respektive metodiky jako SCRUP, OpenUP, extreme programing atd. potlačují formality a věnují se důležitým věcem. Ale samozřejmě ne na úkor disciplíny a pořádku.

Efektivita jednotlivých metodik se měří vcelku jednoduše. A to podle času jaký museli zainteresovaní lidé strávit nad projektem. Což svědčí o kvalitě techniky. Většina metodik se snaží v týmu pracovníků nastolit taková pravidla, aby práce byla co nejefektivnější a spolupráce nevázla. To je docíleno také tím, že množství procesů je omezeno na potřebné minimum tak aby metodika nebyla matoucí a přesto efektivní a přínosná.

Agilní metody vývoje se orientují na rozdíl od tradičních metod na co největší počet iterací. Tudíž délka iterací se snižuje na minimum a snaží se dát zákazníkovi prostor vyjádřit se, zda vyvíjený produkt je dle jeho představ a případně doplnit jiná (nová) specifika. Tento přístup posouvá na rozdíl od tradičních metod analýzu a specifikaci na pozadí. Protože zpětná vazba od klienta je dostatečná a klient upřesňuje své požadavky, na základě funkčního produktu dané iterace. Tím pádem rozsáhlá analýza není důležitá a většině zákazníku nic neřekne objektovému návrhu nebo části návrhu databáze. Zákazník na základě produktu odsouhlasí, zda-li je to dle jeho představ nebo přímo dospecifikuje své požadavky a tato dospecifikace se provede v další iteraci.

Větší počet iterací však způsobuje větší potřebu testování. Některé agilní metody dříve specifikují testy, kterými software musí projít a ne test na software. A to z důvodu aby se netestovaly požadavky podle implementace, ale požadavky podle zadání. Jelikož agilní metody obsahují hodně iterací. Často se neklade takový důraz na samotnou dokumentaci. Avšak kód se tvoří podle sady pravidel. Tak, ať je dobře čitelný a tudíž může vývojáři sloužit jako dokumentace.

	Tradiční metody	Agilní metody
Plánování před implementací	Absolutní	Minimální
Iterace	Minimální, časové náročné	Časté a velmi krátké
Velikost týmu	Obvykle velký	Malý a často kreativní
Kladení důrazu	Na proces	Na lidi
Dokumentace	Důležitá	Minimální
Vedení	Vertikální řízení	Příkazy a kontrola
Velikost projektu	Velké	Malé
Měření úspěchu projektu	Porovnání specifikace a projektu	Spokojenost zákazníka
Přístup	Předem znám	Adaptivní podle podmínek
Funkcionalita	Fixní	Proměnná
Čas a lidské zdroje	Proměnný	Fixní

Obrázek 2: Srovnání tradičních a agilních metod

4. OpenUp

4.1. Základní informace

OpenUp je opensourcový proces, který má na starost EclipseFoundatio. OpenUp vychází z RUP a zachovává jeho základní rysy jako je: iterativní vývoj, scénáře užití, management rizik a přístup podle architektury. OpenUp by se dal nazvat jako sadou best practices vývoje software od počátku až po údržbu.

Iterativní přístup znamená, že se provádí více iterací v rámci celého životního cyklu. Pomocí tohoto přístupu klient vidí velmi často funkční verzi software. Díky tomu je možné se rozhodovat značně rychle a nevznikají dlouhé prostoje. A velice snadno je klient schopen řídit

projekt tak, aby ho orientoval požadovaným směrem. Tímto přístupem také minimalizujeme analýzy. Což díky krátkým iteracím, není překážkou. Mnoho části bylo OpenUp z RUP zachováno, mnoho však jich bylo také vyjmuto a některá byla sloučena tak, aby bylo dosaženo jednoduchosti.

OpenUP je popsáno tak, aby bylo zcela jasné, jak jej nasadit. A to v 3 variantách.

Minimální – jen potřebné minimum jako je.

- Iterativní vývoj
- Spolupráce v rámci týmu
- Neustálá integrace a testování
- Časté vydáváni funkční verze pro klienta

Kompletní – je to hlavní proces vývoje. Oproti minimálnímu je rozšířen o:

- Tetováním systému podle zákazníkových požadavků.
- Prací se určují priority. Na základě těchto priorit jsou jednotlivé úkoly přebírány pracovníky a zpracovávány.
- Snaha zjistit požadavky zákazníka.

Rozšířený – OpenUp je základním procesem. Rozšiřujeme jej o:

- Nové role
- Přidání kroků a úkolů
- Změny a rozšíření nových životních cyklů.

4.1.1. Kde nasadit openUp?

Všude tam kde jednotliví spolupracovníci sdílejí prostor. Například společně sedí v kanceláři nebo OpenSpace místnosti. Velikost týmu, jenž používá OpenUp, by měla být cirka 3 až 6 členů. Z tohoto vyplývá, že openUp je vhodný pro malý počet lidí. Skupinka musí mezi sebou často komunikovat, což je do jisté míry zajištěno sdílením společných prostor. V týmu najdeme následující role: stakeholder, vývojář, architekt, projektový manažer a tester (tyto role jsou dále popsány níže). Každý člen týmu provádí své vlastní rozhodnutí na základě toho, co vykonává a na prioritách, které jsou v projektu určeny - vše tak, aby byla naplněna vize stakeholdrů. U openUp se počítá s relativně krátkou dobou vývoje; cca 3 až 6 měsíců.

Jednotliví členové naleznou v OpenUp definici své role. Definice se skládá ze sady artefaktů a aktivit, za které má člen zodpovědnost.

Zákazníci právě díky OpenUp ví, co mohou očekávat od vývojového týmu a jakým způsobem bude probíhat vývoj software. V openUp je popis co musí stakeholdři dodat vývojovému týmu a jak mu mohou být nápomoci při vývoji tak aby výsledný produkt splňoval jejich vizi.

4.2. Slovníček pojmů

Work produkt – jsou to dokumenty (vize, plány, specifikace projektu…), role, modely (use case, návrhy tříd, návrhy subsystému), databáze, zdrojové kódy atd.. Výsledný produkt dělíme na dvě části. Artefakt, výstup a skupinu výstupu. Artefakt pro konkrétní věc a výstup (outcomers) pro věci abstraktní. Skupina výstupu (deliverables) pokud jde o balík artefaktů.

Role – definuje každému jednotlivci v týmu jeho chování. A to jak k jeho nejbližším spolupracovníkům, tak k celé skupině. Role popisuje odpovědnost pracovníka a nepopisuje daného pracovníka. To znamená, že jeden pracovník může mít více roli a jedna role může mít také více pracovníku. Role se v průběhu prací na projektu mohou měnit u jednotlivých pracovníků. Role programátor, analytik, tester. Jak již bylo popsáno výše, v openUp máme několik základních rolí:

- **Analytik** – je to pracovník, jenž stojí mezi programátorem a klientem. Jeho úkolem je získávat informace o projektu a převádět je do vhodné formy. Analytik vytváří tzv. analýzy jednotlivých procesů, slovník pojmů a vize.
- **Architekt** – stará se o definici zvolené architektury. Vytvořená definice musí obsahovat základní technické parametry a jednotlivá omezení návrhu. Za vše s tímto spojené, je odpovědný právě tento člen týmu.
- **Programátor** – vychází z architektury. Jeho hlavním úkolem je vyvíjet část projektu, za kterou je zodpovědný. Vytváří tak United-testy a stará se o integraci komponentů jak 3. stran, tak vnitřních.
- **Projektový manažer** – je to člověk, který vede projekt do jeho zdárného cíle. Snaží se dopomoci s komunikací klienta. Je zodpovědný za finální projekt, ale také například připravuje jednotlivé iterace včetně jejich rozsahu a presentace jednotlivých nebo vybraných iterací klientovi.

- **Klient** – je to jeden nebo více lidí, jež požadují software. Je to ve své podstatě klasický zákazník, jehož potřeby se snažíme uspokojit. V open Up je i označení stakeholder a to se dá přirovnat k roli klienta.
- **Jakákoliv role** – může vznést požadavek na úpravu. A může vykonávat elementární činnost, jako je hodnocení, pracovat na jednotlivých iteracích atd.

Úkol – je to práce, kterou vykonal někdo v dané roli. Je to série kroků, která vede k nějakému nastavenému cíli.

Proces – proces je sjednocení mezi Úkolem, rolí a produktem. K tomuto sjednocení přidává strukturu a popis postupu. Jednotlivé úkoly pak označujeme jako plánování, provedení atd.. A můžeme je uskutečnit nebo jakkoliv obstarat.

Artefakty - v OpenUp máme mnoho artefaktů, zde bude popis několika nejdůležitějších z nich.

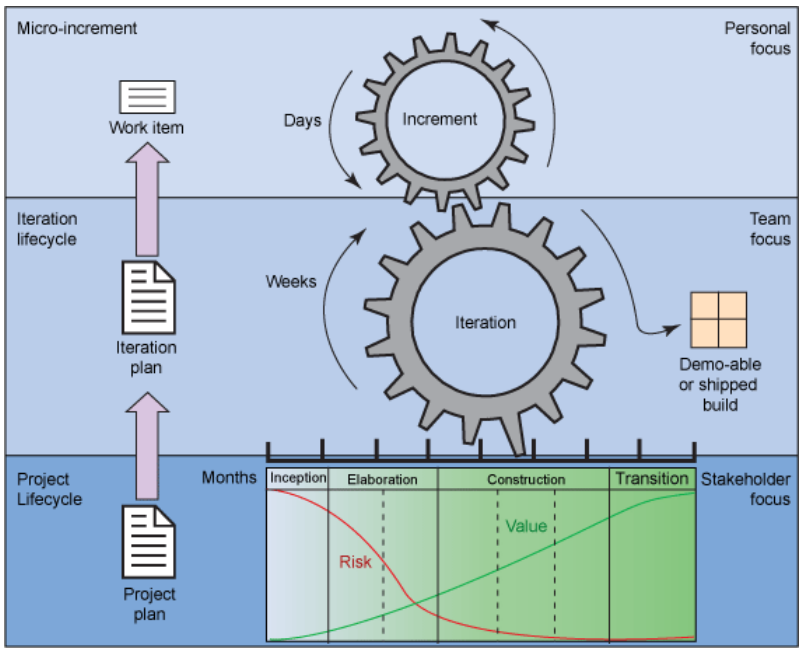
- Projektový plán – veškeré významné informace, které jsou důležité pro strategické rozhodnutí
- Iterační plán – obsahuje seznam cílů jednotlivých iterací. Dále kdo má přiřazenou jakou práci a plán, jak jednotlivé iterace zhodnotit, zda byla úspěšná či nikoliv.
- Implementace – jsou to soubory, z kterých se skládá celý projekt. Jedná se o zdrojové soubory, databázové tabulky, kompilační skripty, nápovědy atd..
- Poznámky k architektuře – Obsahuje omezení v rámci architektury, jaké byly důvody pro tato omezení, a další seznamy rozhodnutí architekta.
- Build – jednotlivá iterace včetně zdrojových souborů.
- Seznam úkolů – soupis veškerých úkolů, které musí být na projektu provedeny.
- Slovník – klasický slovník všech důležitých pojmů
- Vize – je to představa zákazníka (stakeholdra) výsledného produktu.
- Případy užití – Use case, jsou definice chování systému na jednotlivé akce
- Technická specifikace – je to ucelený soubor, který se skládá z těchto artefaktů:slovník, Systém-WideRequirments, Use case a vize.
- Seznam rizik – rizika, která jsou spojena s projektem
- Mezi zbývající artefakty patří hlavně věci okolo testování (test case, test log, Test skript, developer test).

4.3. Vrstvy OpenUp

OpenUp obsahuje několik základní vrstev. Na tomto místě si je popíšeme.

- **Micro-Increments** – Nejnižší vrstva. Jsou to základní velice krátké úkoly. Ty jsou neustále vykonávány a dá se pomocí nich změřit pokrok ve zpracovávání projektu. Díky micro-increments získáváme velice brzy zpětnou vazbu a je nám umožněno adaptivní řízení včetně řízení během iterací.
- **Lifecycle iterativ** – struktura je velice podobná jako u micro-increments. Avšak na jejich konci vychází stabilní build. Který je předváděn zákazníkovi. Jednotlivé iterace jsou řádově v týdnech oproti velmi krátkým inkrementacím z předchozího bodu.
- **Project lifecycle** –se dále dělí na:
    1. Počáteční
    2. Upřesňovací
    3. Konstrukční
    4. Předávací

    Dobrý dohled je zajištěn tím, že celý životní cyklus vývoje je přehledný jak pro zákazníky, tak pro členy týmu. Tyto zúčastněné osoby mohou také dělat rozhodnutí. Projektový plán pak dále definuje: životní cyklus, konečný výsledek a vydanou aplikaci.



Obrázek 3: Vrstvy OpenUp

4.5. Základní princip OpenUp

OpenUp stojí na čtyřech základních pilířích. Tyto pilíře jsou navzájem principiálně propojeny. Je nutné, aby výsledný projekt byl co nejvíce přínosný pro stakeholdra. Ale výsledný produkt musí být vyvážením mezi kritickými a cenově přínosnými kompromisy, úpravami a dodatečně navrženými změnami. Nalezení takové rovnováhy je výzvou a to z mnoha důvodů, jako například dynamických změn a velkého počtu míst.

V době vývoje se objevují stále nová kritická místa a zároveň stará kritická místa jsou zažehnána. Vývojový tým neustále zjišťuje nové skutečnosti o systému a navzájem je sdílí. Jelikož se systém takto dramaticky mění a objevují se stále nové kritické místa, musí být, jak členové týmu tak stakeholdři, připraveni na přehodnocení závazku, priorit a očekávání ohledně výsledného produktu.

Aby bylo možné dosáhnout toho, že systém bude maximálně korespondovat s vizí stakeholdrů, musí všichni zúčastnění bezpodmínečně porozumět veškerým požadavkům stakeholdra. Obzvláště je nutné, aby pochopili následující body.

- Co je nutné provést k vyřešení problému
- Jaká omezení má vývojový tým (čas, úkoly, zdroje, řízení).
- Omezení, které má projekt (cena, čas)

4.5.1. Spolupráce, jež je založena na porozumění a společných zájmech

Ve vývojovém týmu jsou lidé mnoha zkušeností a dovednosti. Je nezbytné, aby tito lidé byli schopni pracovat společně, dokázali se navzájem doplňovat a aby se společné úsilí stalo dostatečně efektivní.

Pokud ve vývojovém týmu vládne pozitivní nálada, je velice efektivní a získává tak dobré zkušenosti pro budoucí práci. Pozitivní náladu navodí zkušení pracovníci, kteří mají hojné zkušenosti z minulosti a dokážou případně poradit a sdílet své zkušenosti.

4.5.2. Výběr/vývoj architektury

Architektura software je popis, jak uspořádat jednotlivé komponenty softwaru. Jaké rozhraní software bude obsahovat. Popis, jak se pomocí těchto rozhraní bude komunikovat. Architektura tedy slouží k celkovému popisu vyvíjeného softwaru a umožní nám vyvíjet software efektivně. Je to jakoby stavební plán, který určuje základní aspekty, ale nezabývá se

detaily. Architektura, krom toho, že umožní efektivní vývoj software, také umožní budoucí snazší rozšíření, předělání nebo integraci do jiného systému.

4.5.3. Zpětná vazba a inovace

Již ve velmi rané části je možné u OpenUp předvádět funkční verze stakeholdrům a pak získávat zpětnou vazbu.

Jelikož během úvodních rozhovorů se stakeholdrem není možné stanovit veškeré požadavky a vize. Proto je nezbytné předvádět vývojové verze tak, abychom získali zpětnou vazbu od stakeholdra a ujišťovali se tím, že vývoj jde správným směrem, nebo směr vývoje dodatečně koordinovali.

Vývojové verze také presentujeme celému vývojovému týmu, neboť není možné, aby všichni členové týmu byli obeznámeni s funkcí celého systému. Proto jsou potřebné tyto presentace, aby celý tým byl seznámen a zároveň aby mohl připomínkovat jednotlivé části a navrhovat změny a inovace.

4.6. Proces

Proces OpenUp je sjednocení mezi Úkolem, roli a produkt. K tomuto sjednocení přidává strukturu a popis postupu. Jednotlivé úkoly pak označujeme jako plánovaní, provedené atd. V OpenUp nám proces prochází čtyřmi fázemi. Všechny fáze probíhají v iteracích, přičemž se doporučuje minimálně jedna iterace.

1. **Počáteční fáze** – je to počáteční fáze v, které začíná OpenUp. V této fází je sepsáno několik dokumentů. Na těch se podílí následující role: analytik, architekt, projektový manažer a samozřejmě zákazník. V této fázi se požadavky na požadovaný software sepisují běžnou všem srozumitelnou řečí tak, aby výsledný dokument byl všem pochopitelný. Výsledný dokument obsahuje následující artefakty:
    a. **Slovník** – definuje názvosloví, které používá analytik a zákazník. Tak aby nedošlo k nejasnostem. Dokument sepisuje analytik a přebírá za něj zodpovědnost.
    b. **Vize** – Je to pohled na výsledný software. Tak jak si jej představuje zákazník. Výsledný produkt se nesmí příliš lišit od vize zákazníka. Vizi sepisuje analytik. A obsahuje nejdůležitější funkce a požadavky.

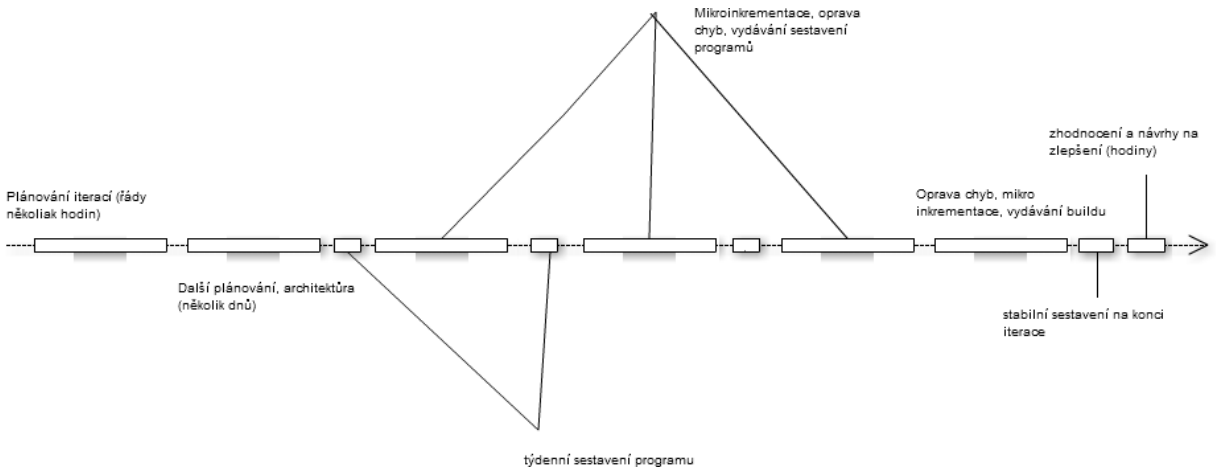
c. **Plán projektu** – popisuje, kdo dostane jakou roli, jaké budou délky iterací. Zhotovuje popisky jednotlivých iterací a vytváří definice jednotlivých iterací. Plán projektu dělá Projektový manažer.

Velice důležité v počáteční fázi jsou schůzky se zákazníkem. Těchto schůzek se účastní analytik, projektový manažer a architekt. Výstupem schůzek je výše jmenovaný dokument. V pozdější fázi, kdy jsou zákazníkovi představovány jednotlivé iterace, jsou také požadavky definovány přesněji, až k naprosto konkrétnímu popisu softwaru.
První, co se dělá v počáteční fázi, je tzv. výkop. Jedná se o získání souhlasu zákazníka s rozběhnutím projektu. Po získání souhlasu se zagituje prvotní představa a vize o budoucím software. V tuto chvíli komunikuje se zákazníkem projektový manažer a analytik.
Celou počáteční fázi má na starost projektový manažer, který se stará o to, aby se projekt posouval správným směrem, komunikuje s členy týmu, se zákazníkem a na základě těchto informací postupuje dále. Případné problémy se snaží vyřešit již v počátku.
Ve chvíli, kdy máme ranou fázi za sebou, přichází fáze, kdy se vybírají a identifikují požadavky. Doplňují se slovníky, rozepisují se use case. Vytvoří se celá technická specifikace projektu. Uvedená specifikace obsahuje také popis požadavků, omezení a cíle, jenž musí výsledný produkt splňovat. Toto vše vytváří analytik. Tester v tuto chvíli vytvoří test case, na základě technické specifikace od analytika. Tudíž do budoucnosti se bude programovat tak, aby konečný software prošel vytvořenými testy a ne s opačnou posloupností, aby se vytvářely testy, sedící na výsledný software.

Jako poslední účastněná osoba v počáteční fázi je architekt. Ten na základě technické specifikace definuje architekturu a artefaktu a poznámky k architektuře. Na konci architektovy práce již vidíme základy budoucí architektury.

Pokud to tedy shrneme; na konci počáteční fáze máme ujasněno, jaké software budeme vyvíjet. Chápeme základní funkcionalitu, navrhujeme možná řešení. Přinejmenším, alespoň jedno. A na základě těchto informací stanovíme předběžný odhad ceny.

2. **Fáze upřesnění –** V této fázi již všichni pracují, programátoři na základě hrubého nastínění architektury z předchozího kroku. Analytici stále upřesňují detaily vyvíjeného díla. Architekt detailně zapracovává specifikaci architektury a v případě naprogramování konkrétné části tester může testovat. Projektový manažer detailně plánuje další iterace a upřesňuje cenu na základě informací od ostatních.



Obrázek 4: Průběh konstrukční fáze

Výsledkem tohoto kroku je upřesněná analýza architektury. Část architektury je již naimplementována a otestována. Tudíž je validována. Zákazníkovi je detailněji představen cenový návrh.

3. Konstrukce

V této fázi se provádí samotná konstrukce neboli vývoj software. Programátoři pokračují na jednotlivých iteracích a zpracovávají jeden úkol za druhým. Analytik ve stejné chvíli získává zpřesňující informace a nové požadavky od klienta. Projektový manažer řídí jednotlivé iterace a plánuje další. Počátkem konstrukční fáze by měla být již hotová validována architektura. To nám zajistí, že mnoho počátečních problémů a rizik je dávno za námi. Projektový manažer se tudíž může věnovat organizování práce a vyjednávání se stakeholdry o výsledné ceně.

Ve chvíli kdy programátor dokončí nějaký úkol tak se vytvoří test a tester jej otestuje. Výstupem práce testera je log o provedeném testu, který se předává k dalšímu zpracování programátorovi. Takto probíhají iterace, dokud je daná testovaná iterace bezchybná.

Jednu iteraci konstrukční fáze můžeme vidět na obrázku č.4. Každá iterace začíná krátkou poradou, kdy se probere naplánování. Tato porada trvá jen několik hodin. V následujícím kroku, který setrvává den, až dva se probírá, jak seřadit úkoly, které je nutno provést v rámci iterace. Také se zjišťuje, jaké dopady bude mít na iteraci předešlou. Nejdelší část iterace zabere provádění naplánovaných úkolů. Tomuto dílu se říká mikro inkrementace. Na konci každé této mikro inkrementace by měl vyjít

otestovaný úkol. Na konci týdne by tedy měl být k dispozici stabilní build. Obzvlášť velká pečlivost se pak klade buildum, které se předvádí zákazníkům. V poslední části iterace, většinou týden až dva, se dbá na testování a opravy chyb. V této fázi se můžeme dobrat k tomu, že je potřeba doplnit některou z funkcionalit, popřípadě výjimečně vytvořit úplně novou. Ke konci je pak sezení s stakeholdry a probíhá hodnocení vydaného buildu, jak se vyvíjel, a navrhnou se podle získaných zkušeností případné zlepšení pro další iteraci.

Je vhodnější, pokud se jednotliví členové týmu rozhodnou, jak budou plnit své závazky. Tato skutečnost motivuje lidi více, než skutečnost, že je jim postup jejich práci pevně určen, jelikož každý člen týmu bývá lepší v nějaké konkrétní činnosti. To má za to, že každý člen týmu si vezme vyhovující úkol. Pokud je tato organizace práce zvolena, je nutné, přizpůsobit plánování práce, kontrolu plánování závazku a je nutné zjistit od jednotlivých členů, jak vidí svou roli v rámci projektu.

Pro tzv. fungování samo-organizace je podstatné, aby fungovaly dva základní principy.

a.Transparentnost – pokud je všem jasné, kdo bude plnit jaké závazky, bude práce v týmu efektivnější. Jedná se zejména o závazky v rámci iterace popřípadě projektu. Komunikace musí být otevřená. Díky tomu se problému ujmou ti správní lidé.

b. Coaching – v OpenUp je coachem projektový manažer. Projektový manažer musí splňovat několik vlastností. Hlavní z nich je command-and-control řízení. Tento způsob řízení je znám již 2000 let. Hlavní využití našel v armádě.

Zakončením konstrukční fáze je beta verze výsledného programu.

4. Předávací fáze – po konstrukční fázi a všech potřebných testování nastává poslední fáze - předávání. V této fázi se předává výsledný projekt stakeholdrům. Provádí se testování, které je dle use case. Testováním se zajistí to, aby výsledný produkt splňoval požadavky zákazníka. Stakeholdři připomínkují výsledný produkt a tím se snažíme doladit funkcionalitu, zlepšit výkon a kvalitu.

5. SCRUM

Je to asi neznámější agilní metoda vývoje, které je vhodná pro týmy velikosti od 4 do 15 lidí. Tato metoda víceméně počítá s tím, že tým sedí v jedné místnosti. Ovšem najdou se i případy, kdy se používá napříč světem. SCRUM navrhli v roce 1986 pánové Hirotaja Takeuchi a

Ikujiro Nonaka. Metodika se však rozšířila až na začátku 90. let hlavně díky pánům Ken Schwaber a Jeff Sutherland, kteří nastavili délku iterace na 30 dní a vpravili backlog a další artefakty. V SCRUMU jde o to, že jednotlivé fáze projektu se překrývají a celý proces stvoří jeden multifunkční tým. Tým je multifunkční z toho důvodu, že během různých fází vývoje plní různé úkoly. Na anglické wikipedii je SCRUM multifunkční tým přirovnáván k rugby, hlavně proto, že se celý tým snaží překonat hřiště jako celek. K překonání využívá přihrávek vpřed i vzad. A následná tlačenice, která je při opětovném zahájení hry, je přirovnána k SCRUM.V překladu totiž scrum znamená tlačenice nebo skrumáž.

Obrázek 5:Scrum

V popisu SCRUMU stojí jako hlavní body; Iterativní, inkrementální framework pro managment kompletních prací. Jelikož jde víceméně o framework nezabývá se SCRUm kompletním popisem, jak má být co uděláno. V SCRUMU se spoléháme na inteligenci a to, že samotný tým ví, jak se co má udělat nejlépe.

5.1. Role účastníků Scrumu

V scrumu fungují dvě role, scrum master and product owner. Scrum Master je takový kouč týmu, jehož úkolem je pomáhat členům týmu a dodržování SCRUM frameworku. Product owner je v SCRUMU role zákazníka, který pomáhá celému týmu s vývojem kvalitního produktu.

Níže je uveden vtip o praseti a kuřeti. Prase musí dát do projektu maso a kuře pouze vejce. Prase je tedy zapojeno do projektu a kuřete se problém pouze týká.

- Pigs (osoby, které přímo souvisejí s vývojem aplikace)
- Chickens (uživatelé produktu, manažeři, kteří přímo nezodpovídají za vývoj)

Obrázek 6:Vtip o Scrum

Mezi Pigs patří tyto role:

- Product Owner je osoba, která zodpovídá za priority, za to, co se bude v příštím sprintu implementovat a určuje implementační detaily. Je také odpovědná za ziskovost projektu. User stories přiřazuje priority a obchodní hodnotu, upravuje položky mezi jednotlivými Sprinty a schvaluje mimo jiné jednotlivé výsledky programátorů.
  Product Owner je osoba, která je naprosto obeznámena s veškerými aspekty projektu. A pokud bude mít člen týmu jakýkoliv dotaz k projektu, musí být projekt owner schopen na něj odpovědět. Je to mezičlánek mezi zákazníkem a realizační části. Často tuto roli zastává někdo z marketingové části firmy.
- Scrum Master pracuje jako ten, kdo má programátory odstínit od okolního světa. Řídí vývojáře, ale zároveň se stará o to, aby jim fungovaly počítače, měli dostupný software, který potřebují, řeší spory, kontroluje vytíženost jednotlivých programátorů tak, aby jejich čas byl využit efektivně. Scrum Master a manažer se mohou krýt, zároveň může být Scrum Master i analytik. Je ale zakázáno, aby byl Scrum Master zároveň programátor, v takovou chvíli by nebyl schopen odfiltrovat programátory od rušivých vlivů – byl by sám zranitelný. Pro tuto roli se jednoznačně hodí Project Manager. Scrum master také podporuje spolupráci mezi všemi rolemi a funkcemi. I když se Scrum master stará o celý tým, nepřiděluje jim jednotlivé úkoly. Ty si dělí

samotný tým. Je velice důležité, aby Scum Mater byl přítomen na pracovišti. Jedině tak může svou práci dělat kvalitně.

- Scrum tým – Je to tým lidí, o které se stará Scrum Master. Tým, který vytváří hodnoty pro zákazníka a je schopen zastat mnoho funkcí. Jak již bylo řečeno výše, počet členů je mezi 5 až 9 a organizuje sám sebe. Tým má povoleno dělat vše proto, aby byl splněný cíl. Členové se nedělí na programátory, databázisty, testery atd. Každý si je v Scrum týmu roven. Tím vzniká bezkonfliktní, kamarádské prostředí. Členové Scrum týmu se mění jen mezi jednotlivými sprinty.

Mezi Chickens patří následující role:

- Stakeholders – lidé od zákazníka, testeři, připomínky zvenčí
- Managers – osoby, které pomáhají nastavit prostředí, ale nejsou ani vývojáři, ani Product Owneři, ani Scrum Masteři

U SCRUMu je velice vhodné a často i aplikované, že je u vývoje účastněn i zákazník. Díky tomu může být při diskuzích o vývoji a neustále připomínkovat tak, aby výsledný produkt byl dle jeho představ.

Před dalším popisem Scrumu si musíme objasnit dva důležité pojmy.

- **Překážka** – je to cokoliv co zabraňuje, jak jednotlivci tam týmu, pokračovat nebo efektivně vykonávat práci.
- **Abnormal Termiation –** je nečekané přerušení sprintu. Po přerušení sprintu znovu probíhá Sprint Plaining Meting. Sprint se může přerušit z několika důvodu.
    - Tým došel ke skutečnosti, že většinu naplánovaných úkolů nestihne.
    - Managment přeruší sprint z důvodů neplnění úkolu.
- **Sprint –** v openUp pojmenováno jako jedna iterace.

5.2. Artefakty

- **Product backlog –** Kompletní specifikace toho co se bude vyrábět. Na obsahu se podílí kdokoliv. Hrubé odhady, které jsou uvedeny v backlogu, jsou oceněny bodovou stupnicí. V budoucích iteracích se pak zjišťuje, jak moc úsilí tyto body znamenají a

nadále se nacenění zpřesňuje. Product backlog obsahuje celkový popis veškerých požadavků na výsledný systém. Požadavky se řadí sestupně dle priority.

	Item #	Description	Est	By
Very High				
	1	**Finish database versioning**	16	KH
	2	**Get rid of unneeded shared Java in database**	8	KH
	-	**Add licensing**	-	-
	3	Concurrent user licensing	16	TG
	4	Demo / Eval licensing	16	TG
		Analysis Manager		
	5	File formats we support are out of date	160	TG
	6	Round-trip Analyses	250	MC
High				
	-	**Enforce unique names**	-	-
	7	In main application	24	KH
	8	In import	24	AM
	-	**Admin Program**	-	-
	9	Delete users	4	JM
	-	**Analysis Manager**	-	-
	10	When items are removed from an analysis, they should show up again in the pick list in lower 1/2 of the analysis tab	8	TG
	-	**Query**	-	-
	11	Support for wildcards when searching	16	T&A
	12	Sorting of number attributes to handle negative numbers	16	T&A
	13	Horizontal scrolling	12	T&A
	-	**Population Genetics**	-	-
	14	Frequency Manager	400	T&M
	15	Query Tool	400	T&M
	16	Additional Editors (which ones)	240	T&M
	17	Study Variable Manager	240	T&M
	18	Haplotypes	320	T&M
	19	**Add icons for v1.1 or 2.0**	-	-
	-	**Pedigree Manager**	-	-
	20	Validate Derived kindred	4	KH
Medium				
	-	**Explorer**	-	-
	21	Launch tab synchronization (only show queries/analyses for logged in users)	8	T&A
	22	Delete settings (?)	4	T&A

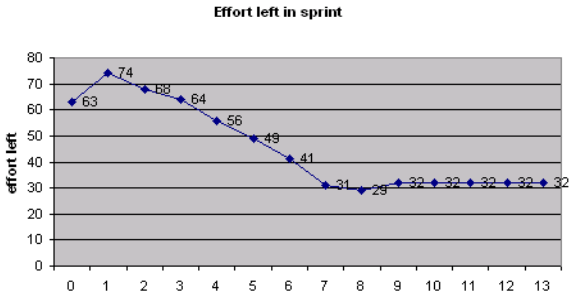
Obrázek 7: Ukázka jak může vypadat product backlog

- **Sprint backlog –**je podmnožina product backlogu. To znamená, že obsahuje vybrané body z product backlogu, které se v daném sprintu budou provádět.

Sprint 3. Plug in the Real Weather

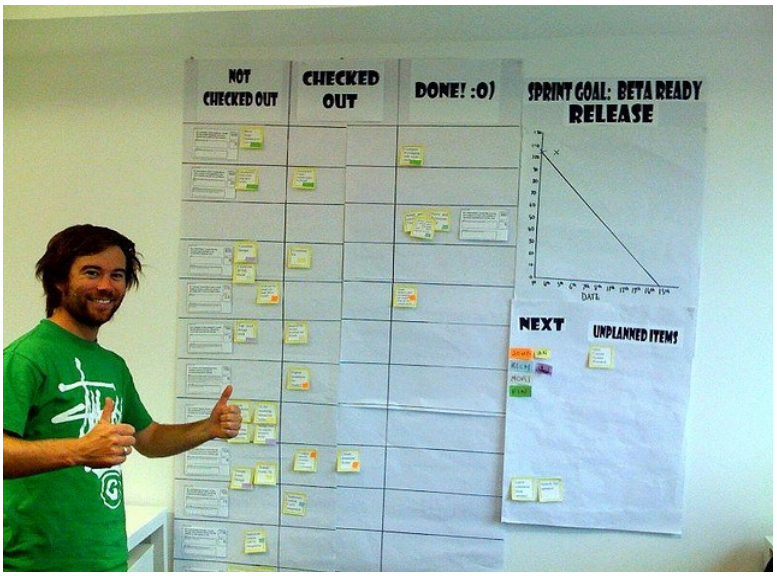
Story ID	Story/task	days in sprint / effort left												
		0	1	2	3	4	5	6	7	8	9	10	11	12	13
		63	74	68	64	56	49	41	31	29	32	32	32	32	32
10	**Fetch one day temperature data from the weather provider system**														
	Make our server connect and authenticate to the provider system	4	4	1								3	3	3	3
	Read provider's data directory	8	7									0	0	0	0
	Parse the current temperature out of the data	6	6									1	1	1	1
	Push the temperature data to the client	16	16	1								0	0	0	0
11	**Fetch rain, snow, etc details from the provider**														
	Parse snow/rain data from the provider's data	4	4									0	0	0	0
	Push the snow/rain data to the client	4	4									0	0	0	0
	Redesign client screen a bit											3	3	3	3
	Refactor the server code										4	4	4	4	4
12	**Fetch several days data from the provider**														
	Parse the weather data in day packs	10	10	10	10	10	10	10	10	10	10	10	10	10	10
	Push several days data to the client	3	3	3	3	3	3	3	3	3	3	3	3	3	3
13	**Auto-refresh feature**														
	Make the client ping server once per 4 hours	6	6	6	6	6	6	6	6	6	6	6	6	6	6
	Make the server update the client	2	2	2	2	2	2	2	2	2	2	2	2	2	2

The team underestimated the difficulties of connecting to the provider's CORBA server. Even though some work has been already done, the remaining amount is much bigger, than the original estimation for the whole task



Obrázek 8:Ukázka jak může vypadat product backlog

- **Burn down – sprint burn down** chart je tabulka, která je všem přístupná. A je v ní obsaženo, co je nutné dodělat v daném sprintu. Tabulka se aktualizuje nejlépe po dokončení jakéhokoliv úkolu. V horším případě se aktualizuje jednou denně.



Obrázek 9:SCRUM v Praxi

5.3. Schůzky

- **Daily scrum –** Je to krátká (15min) každodenní schůzka, která je většinou na začátku pracovní doby v pevně určený čas. Většinou se účastí jen "pigs". Každý člen týmu veřejně přednese, co udělal od poslední schůzky, tedy posledního pracovního dne, a co plánuje na tento pracovní den. Případně se zmíní, že mu bráni něco k provedení práce. Konkrétně se pak probírá se Scrum Mastrem jak tyto problémy vyřešit a to již mimo daily scrum, tak, aby nebyli zdržováni ostatní členové schůzky.
- **Scrum of Scrum –** Pokud na projektu pracuje vice scrum týmu, musí proběhnout schůzka jednotlivých týmu. Za každý tým, se tedy vyšle 1 člověk a ti se setkají na Scrum of Scrum schůzce. Scrum of Scrum probíhá vzápětí po Daily scrum. Otázky, jež se probírají, jsou velice podobné jako u Daily scrum, jen se ptáme, co celý scrum udělal od poslední scůzky a jaká je naplánována práce na dnešní den. Zda má Scrum nějaký problém, který mu braní v postupu a jestli nehodlá narušit práci nějakému jinému scrumu svým zásahem.
- **Sprint plaining Meeting –** Opět porada před začátkem tentokráte sprintu. Tato porada je v rozsahu jedno pracovního dne a její náplni je výběr práce na následující sprint. Vytvoří se tzv. Sprint backlog. To je soubor, jenž podrobně popisuje kolik času zaberou jednotlivé úkoly.

Tasks	Mon	Tues	Wed	Thurs	Fri
Code the user interface	8	4	8		
Code the middle tier	16	12	10	4	
Test the middle tier	8	16	16	11	8
Write online help	12				
Write the foo class	8	8	8	8	8
Add error logging			8	4	

Obrázek 10:Sprint plaining meeting

- **Sprint Review meeting –** oproti sprint plaining meeting je tato porada na konci sprintu. Délka je přibližně na 4h. Provádí se zhodnocení spintu a práce které se stihla či nikoliv. Na konci Sprintu se také předvádí dílo stakeholdrům.

- **Sprint retrospective –** opět meeting na konci sprintu se všemi členy týmu. Jednotliví členové zhodnocují právě provedené sprinty. Co jsem jim povedlo či nepovedlo a co by případně příště šlo udělat lépe.

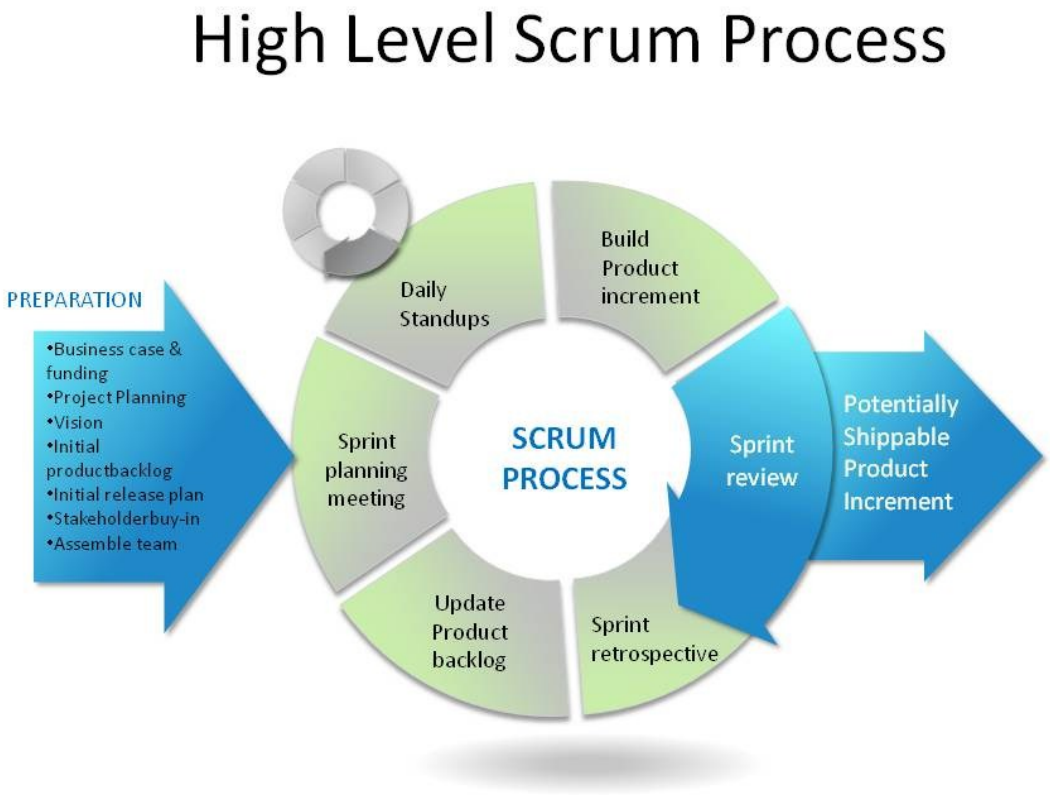
5.4. Proces v Scrum

Nyní již známe všechny potřebné role a detaily Scrumu a popíšeme si samotný proces. Na začátku každé iterace se provede Sprint plaining meeting. Zde product owner nastaví jednotlivé user sories priority. Každý tým si nyní vybere ty úkoly, na kterých by chtěl pracovat. Poté jsou user storie přesunuti do dalšího kroku Sprint Backlog. V sprint backglogu pak dále probíhá rozepisování do dalších dílčích úkonů a provádí se jejich bodové oceňování.

Scrum zakazuje přesčasy a považuje práci za kvalitně odvedenou pouze v případě odpočatého programátora. Proto se nastavují, tzv Time Boxy což je termín, v kterém má být danný sprint dokončen. V Time Boxu se termín odevzdání a dokončení práce kryje. Což znamená, že se nepočítá s jakoukoliv rezervou a také se nijak neřeší, pokud tým udělá svou práci dříve.

Zákazník nemá pravomoc zasahovat do jednotlivých sprintů a určovat priority nebo měnit jejich obsah. Na to má oprávnění jen mezi jednotlivými spritny. Pokud by to nebylo dodrženo, Time Box by postrádal smysl.

Jak již bylo řečeno , každý den probíhají Saily Scrum. Na konci sprintu je pak hotový produkt, který obsahuje kompletní funkcionalitu, která byla naplánována na danný sprint. Tento výsledek je pak představen na Sprint Review Meetingu. Produkt se pak porovnává s backlogem. A pokud je vše, co bylo naplánováno v backlogu splněno, produkt se předává zákazníkovi, nebo v opačném případě se splňuje další sprint.



Obrázek 8:SCRUM

6. Extremní programování

5.2. Základní informace

Veřejnosti asi nejznámější agilní metodika. Extremní programování vzniklo v 90. letech ve skupině lidí okolo Kenta Becka. Kent Beck je chápán jako hlavní autor extremního programování a napsal asi i nejdůležitější knihu "Extreme programming Explained". Extremní programování (XP) určuje činnosti všem členům týmu. Tyto funkce jsou právě vyvedeny až do extrému. Toto vyvedení tradičních činností do extrému by mělo být výhodné v podmínkách kdy se zákazníkovo zadáni velice často mění.

XP je vhodné pro malé, až střední týmy. XP je vlastně sada „best practises" založených na inkrementálním vývoji. Ptáte se, jaký je XP rozdíl oproti klasickým sadám best practies?

- Již od první verze je projekt spustitelný
- Ve chvíli kdy se nějaká revize zdrojového kódu osvědčí, pak je neustále revidována.

- U jednoho počítače sedí 2 programátoři, kteří se střídají. To zabraňuje profesionální slepotě. Střídání programátorů je neustálé. Vyměňují si klávesnici dle kreativního nápadu.
- Ve chvíli kdy se osvědčí testování, se začne testovat naprosto vše a neustále. Testuje jak programátor za pomocí jednotkového testu, tak pomocí akceptačních testů zákazník. V XP se stává, že testovací kód svým rozsahem předčí samotný vyvíjený produkt.
- Opět pokud se osvědčí navrhování kódu tak poté se navrhuje úplně vše. A za pomocí refaktorizace se můžou všichni podílet na návrhu kódu a jeho zlepšení.
- Pokud se osvědčí jednoduchý kód tak jej tak udržujeme. V XP necháváme kód ve funkčním minimu. Čím jednodušší, tím lepší.
- Pokud se potvrdí architektura, tak se všichni podílejí na jejím vývoji.
- Pokud se testuje integrace, následně se testuje integrace veškerých komponentů i několikrát denně.
- Délka iterace je zkrácena na absolutní minimum. Což znamená, že iterační perioda může trvat i minuty. Což potvrzuje, že po naprogramování nějaké části a jejím následném otestováním se integruje do produkční verze.
- U XP se také používá metoda, že nejprve jsou napsány testy a až dle těchto testů je vytvořen samotný kód programu. Psaní testů je ve velice krátkých inkrementacích. Tak se pokračuje, dokud není celý projekt napsán a tím pádem je zajištěno, že je vše otestováno.
- Zaměřujeme se na to, zda výsledný produkt je takový, jaký si objednal zákazník.

U XP se na prvotních schůzkách se zákazníkem sepíše tzv. user stories. Což je popis sada uživatelských příběhů. Na tvorbě user stories se především podílí samotný zákazník. User stories slouží jako základ pro use case. Stories se tvoří se zákazníkem maximálně 2 týdny, načež se spolu s programátorem proberou a určí jejich náročnost. Klient si pak vybere, které příběhy požaduje udělat prioritně. Po jejich dokončení si zákazník vybere další. Dá se říci, že u XP si vývoj řídí samotný klient.

Obrázek 9:Iterace extremního programování

XP se dá zobrazit jako několik vnořených kruhů. Jednotlivé části procesu jsou zpracovávány v různých časových relacích.

- Kód se odehrává ve vteřinách, kdy pár programátorů každou vteřinou vytváří nové a nové kusy zdrojového kódu.
- V rámci minut trvá testování třídy. A to jak jejich tvorba, tak neustále testování.
- Několik hodin denně stráví dvojice programátorů nad tvorbou nového kódu. Jde o tzv. proces párového programování
- Jednou denně si dvojice přiděluje úkoly a to na poradě jménem Stand up meeting
- Plán iterace se vytváří přibližně v měsíčních cyklech. Jelikož jeho zpracování je v řádu týdnu. Iterační plán obsahuje seznam úkolů, jenž je třeba udělat, seznam úkolů, které již jsou udělány a seznam chyb projektu.
- Plán vydání je dokument obsahující seznam úkolů, které je třeba dokončit pro dokončení softwaru. Tento dokument nastiňuje přibližný termín dokončení a předání produktu.

Velká výhoda XP se značí v krátkých inkrementacích. Ty jsou úzce spjaty se zákazníkem a jeho požadavky na výsledný produkt.

6.2. Role

- **XP coach –** podpůrná role, která pomáchá celému týmu v XP dodržovat proces tím, že je zodpovědná za vysvětlení procesu a udržení směru vývoje procesu tím správným směrem. Řeší případné konflikty a vylepšuje a upravuje proces. Pomáhá tím, že přináší zcela jiný pohled na věc. A to z vnější strany na tým. Veškerá rozhodnutí, která XP coach provádí, by měla mít základ na XP jako je komunikace, jednoduchost, odezva nebo odhodlání. XP coach si musí získat respekt celého týmu, aby jeho nařízení byly vykonávány.
- **XP zákazník –** jeho role začíná hned od začátku projektu a to dosti významně sepsáním user stories. Tento 14 denní maratón přinese definice funkcí celého systému. V budoucnu je pak zákazník zodpovědný za zvolení pořadí, kdy se jednotlivé funkce systému budou provádět. Velice důležitým úkonem zákazníka je definování akceptačních testů. Ty se provádějí proto, aby se ukázalo, že systém je spolehlivý a dělá to, co zákazník požaduje. Díky tomu zákazník udělí stanovisko, že produkt je v pořádku a tím jej vlastně převezme.
- **XP programátor –**hlavní pracovní náplní programátora je tvorba kódu. Ten tvoří na základě user storie. Dále zodpovídá za integraci a redaktorování kódu. V každém týmu by měl být zvolen jeden člen jako hlavní programátor. Ten má navíc administrátorské práva. Ty tkví v tom, že se snaží udržet dobrou pracovní morálku a příjemné pracovní prostředí. Administrátor je také designerem, implementátorem, integrátorem a definuje, jaké nástroje na projektu jsou použity.
- **XP tester –** tester provádí testování dle jednotlivých úser stories. Musí se starat o časté testování a pak následné zveřejňování testovacích logů celému týmu. Tester si také musí být schopen vytvořit systém na správu logu. Aby nedošlo k tomu, že se ve velkém počtu logu ztratí, vše musí udržet maximálně přehledné. Tester nemusí znát celý systém do nejmenších detailů, ale musí vyzkoušet veškeré stavy systému, pokusit se vyvolat výjimky a ověřit jejich ošetření. Tester by měl být navíc schopen používat nějaký automatizovaný systém na testy. Velice důležitou vedlejší činností je v XP pomáhat zákazníkovi s definováním akceptačních testů pro jednotlivé user stories.
- **XP tracker -** v překladu stopař. Je komunikačním kanálem mezi zákazníkem a vývojovým týmem. Je schopen říct v jakém stavu je proces vývoje produktu. Tracek sleduje releas plán, iterační plán a jednotlivé výsledky akceptačních testů. Pokud je

tracket dobrý je schopen získat veškeré informace o stavu projektu tak aniž by jakkoliv narušil proces vývoje.

6.3. Artefakty

Počat artefaktů je v XP velmi krátký. Důvod je jednoduchý. Extremní programování samo hlásá velkou jednoduchost. Artefakty, které popíši jsou nejčastěji využívané a nebo se generují v průběhu procesu, a nebo usnadňují práci a vedou k funkčnímu produktu.

- **Vize –** vize je pohled nebo představa zákazníka na to, jak by měl finální produkt vypadat. Přináší týmu vývojářů zákazníkovu představu vyvíjeného produktu. Vize je tvořena těmi nejdůležitějšími body, které by měl projekt obsahovat z pohledu zákazníka. V pozdější fázi se vize používá v nerozhodných etapách, zda chceme něco udělat tak nebo naopak.
- **User story –** vize je několik důležitých bodů, které by měl systém umět a User story je slovní popis těchto funkcí. User story tvoří opět zákazník. Není to kompletní popis všech funkcí systému. Tak jako vize, také neobsahuje veškeré funkčnií body systému. Z user stories se pak s pomocí testera tvoří akceptační testy. Ty slouží k ověření funkčnosti. Tým pak určí, zda je reálné celé user story stihnout v dané iteraci nebo se musí nějak dále dělit na více iterací.
- **Release plán –** poskytuje dlouhodobější pohled na věci, které se budou implementovat. Releas plan se tvoří z user sotorie. Jednotlivé usser stories se zpracovávají v iteracích. Několik iterací pak tvoří release.
- **Iterační plán –** je seznamem jednotlivých user stories. K nim je ještě přidáno několik engineering task. A toto se bude zpracovávat v iteraci. Jednotlivé user stories vybíráme s release plánu. Po dokončení iteračního plánu je iterační plán předělán do fyzické podoby. Například vytištěn nebo pomocí sady papírku vytvořen jako tabule. Jednotlivé úkoly iteračního plánu jsou natolik malé, že každý člen týmu je zpracovává cirka do 2 pracovních dnů. Jelikož existuje tabule s user stories, lze snadno dohledat jednotlivé souvislosti. V případě, že není nějakému týmu zcela jasná funkčnost, nebo souvislost může vývojový tým diskutovat. V případě naprosté bezradností se kontaktuje i samotný zákazník. Na tabuli se po rozdělení úkolů napíše ke každému user stories jméno dvojice, která ji bude zpracovávat. Délka iterace je plánována na délku a určitou množinu user stories. Pokud se v průběhu iterace zjistí, že úkolů je

příliš moc nebo naopak příliš málo, jsou user stories buď přidány, nebo odebrány, tak aby délka iteračního plánu byla dodržena.

- **Jednotkové testy** – jsou to testy, které jsou vytvořeny před samotným naprogramováním produktu. Slouží tedy k ověření funkcionality za pomocí redaktorování a samotnému vývoji zdrojového kódu.

6.4. Základní principy

6.4.1. Komunikace

U projektu je nutno udržovat vysoký standart komunikace. Důvod je prozaický. Pokud selže komunikace mezi zákazníkem a programátorem, může zákazník dostat jiný produkt než si představoval. Pokud selže komunikace na vyšší úrovni tj. mezi manažerem a programátorem může manažer dostávat milné informace, nebo vůbec žádné. A na základě těchto informací může mít manažer ztížené rozhodování. Proto XP má předem dané postupy komunikace tak, aby komunikace byla vyžadována nebo přímo nutná.

- Programujeme v párech
- Používá se v osobní komunikaci běžný slovník
- Upřednostňujeme osobní setkání
- Pracujeme nejlépe v openspace kanceláři
- Obchodní zástupci jsou vždy po ruce.
- Sdílíme kód
- Často informujeme zákazníka o průběhu prací na projektu.

6.4.2. Jednoduchost

V XP programujeme jen aktuální věci, které je třeba provést pro splnění aktuálních požadavků. Nevytváří se nic, co není důležité a tzv. v budoucnu by se mohlo hodit. Udržujeme kód v minimálním funkčním stavu. Další myšlenkou je, že se dnes udělají jednoduché věci a zítra ty složité. Jelikož XP předpokládá, že zákazník je občas kolísavý a přehodnocuje své požadavky, tvorba velkých složitých bloků nemusí být vždy tou správnou cestou. Zmíněné bloky práce můžou být znehodnoceny změnou zadání zákazníka. Kdežto malé části se snadno mohou přepracovat. Tato filozofie platí i o psaní dokumentů. Pokud je nikdo nepotřebuje, nepíšeme je. Při samotném programování se držíme těchto kroků pro to, aby kód byl plně funkční a přitom neobsahoval zbytečnosti.

- Splní všechny požadované testy

- Neobsahuje duplicity
- Musí obsahovat všechny myšlenky a nápady autora
- Musí mít minimalizované třídy a metody

6.4.3. Zpětná vazba

Zpětná vazba na úrovní programátorů se skládá především z jednotkových testů. Ty se píšou před samotným započetím programování projektu. Slouží k tomu, aby programátor měl okamžitou zpětnou vazbu na svůj kód. Tím pádem okamžitě ví, zda jeho kód dělá přesně to, co je po něm vyžadováno.

Velice důležitá je i zpětná vazba od zákazníka. Tomu je umožněno si software prohlédnout každé 2 týdny. Tato zpětná vazba umožňuje zákazníkovi směřovat vývoj produktu požadovaným směrem. Díky připomínkám od klienta máme informaci na to, v jakém stavu se projekt nachází.

Projekt musí procházet akceptačními testy. Ty slouží k tomu, že žádná chyba se v projektu neobjeví dvakrát.

6.4.4. Odvaha

Každý programátor musí mít dost odvahy na to, aby byl schopen se pustit do opravdu velkých úkolů, na většinu úloh totiž navazují další. Takže v případě jeho selhání by mohl selhat celý projekt.

6.5. Praktiky XP

V xtreme progamování máme tři skupiny praktik - business praktiky, týmové praktiky a programovací praktiky. Ty se pak dále dělí do dalších podskupin.

6.5.1. Programovací praktiky

V XP si určí postup práce a priorit zákazník. Proto je důležité, aby to tak zákazník vnímal. Po dokončení programování musí programátor začít s integrací své naprogramované části do výsledného produktu. Tyto integrace by měly probíhat přibližně jedenkrát denně. Po dokončení integrace probíhají testy stabilní verze produktu. Pokud testy neprojdou, může programátor okamžitě zareagovat, protože má v živé paměti celý jím naprogramovaný blok. K sestavování finálních verzí se často používá podpůrný software.

Výhody častých integrací jsou.

- Nevznikají velké chyby, které jsou náročné časově na opravu.

- Zákazník je neustále masírován novými funkcemi, a tudíž vidí pokrok na projektu. Takže je ve skutečnosti mnohem klidnější než když vidí nějakou funkční verzi až na konci projektu.
- Systém v pozdější iteraci je vlastně okamžitě připraven k finálnímu nasazení.

Klíčovou vlastností XP je jednoduchý návrh. Což znamená, že se snažíme pokud možno vytvořit co nejjednodušší software. Tak aby prošel testy a splňoval veškeré požadované vlastnosti. Jednoduchý návrh tedy znamená.

1. Systém projde všemi testy
2. Systém neobsahuje žádný duplicitní kód
3. Z kódu jde poznat programátorova myšlenka.
4. Počet tříd a metod je omezen na minimum

U XP musí programátor na počátku zcela pochopit zákazníkova user stories. Musí chápat co se budě dělat. To je asi nejtěžší část. V XP programingu se nemění již napsaný kód. Měnění může vést k tomu, že programátor nepostřehne veškeré myšlenky předchozího tvůrce kódu. V XP se využívají praktika testů řízených vývojem. Jde o to, že vývoj se řídí pomocí malých testů. Nejprve se utváří testy a pak až se programuje kód. Testy si dělá programátor. A jaká je strategie utváření software v XP?

- V prvních dnech není většinou zákazníkovi zcela jasné, jak software bude po všech stránkách vypadat. Proto je podstatné, aby programátor byl připraven na případné změny.
- Důležitá rozhodnutí se odkládají. Důvod je prostý. V danou chvíli nemusíme znát veškeré aspekty. Proto důležité rozhodnutí odkládáme do doby, kdy je nezbytně nutné je provést.
- I přesto, že důležitá rozhodnutí odkládáme, může přijít chvíle, kdy zjistíme, že v minulosti jsme provedli špatné rozhodnutí. Proto každá část vytvořeného kódu musí být snadno modifikovatelná.
- Je výhodnější namísto návrhu začít přímo programovat. Jelikož přímá tvorba kódu poskytuje mnohem lepší zpětnou vazbu. Málo kdy se totiž stane, že prvotní návrh je natolik dokonalý a přesný, že přežije až do konce projektu.

Praktiky XP, jenž nás učí pracovat jednoduše a vytvářet spravedlivý kód.

- Jelikož důležitá rozhodnutí odkládáme na později, vytváříme v prvopočátku jednoduché úkony. Řešení složitých části kdy je provést důležité rozhodnutí vyplyne z těchto jednoduchých. To, že se nějaké části říká, že je triviální neznamená, že její tvorbu podceníme a nerozmyslíme si veškeré její hlediska.
- Nikdy nepřidáváme nikam nic, co by se mohlo hodit. Pokud to neříká user stories, tak to neděláme.
- Vyvarujeme se duplicit. Jelikož na projektu pracuje více programátorů, pro odstranění duplicit se využívá refaktorizace.

- Je nezbytné si uvědomit, že často je jednodušší smazat celý blok kódu a napsat jej znova, než hodiny dumat nad tím, kde je chyba.

A jaké výhody přináší XP programing obyčejným programátorům?

- Není nutné investovat do frameworků.
- Návrh je primitivní. Z toho vyplývá, že aplikace je tvořena co nejjednodušeji. Tudíž není později problém s přílišnou složitostí a tím pádem problémovými částmi projektu.
- Systém díky své jednoduchosti jde velmi snadno přepsat.

6.5.2. Refaktorování

Definice refektorování zní. „Refaktorování je disciplinovaný proces provádění změn v softwarovém systému takovým způsobem, že nemají vliv na vnější chování kódu, ale vylepšují jeho vnitřní strukturu s minimálním rizikem vnášení chyb.“ To znamená, že v kódu provádíme spoustu malých změn, až kosmetických jako je přejmenování tříd, přesunu kódu atd. Celkový efekt ale refaktorizace je velký. Výsledný kód má pak výhodu v tom, že je velice přehledný a snadno se v něm orientuje a je lehce rozšířitelný. Díky této přehlednosti se snižuje množství chyb, které při dalším vývoji vznikají. A samotný vývoj probíhá velice rychle, protože se programátor značně intuitivně orientuje. Zvláště v XP, kdy pracujeme v týmech je vhodné používat refaktorování z důvodu, že kód je po refkatorování vlastně unifikovaný a nemá tudíž na jeho podobu vliv různý programátorský styl.

Výhody redaktorování –

- Podpora iterativního vývoje.
- Zamezení tvorby nepoužitelného kódu.
- Veškeré budoucí změny bude snazší provést.

Historie – refaktorování se poprvé objevilo v programovacím jazyce smalltalk. Zde existoval první nástroj pro refaktorování. V pozdější době, když již bylo refaktorování dostatečně prověřeno, byla napsána kniha *Refactoring: Improving the Design of Existing Cod.* Ta je do dnešního dne považována za jakousi bibli refaktoringu.

Jak refaktorovat - refaktorování se provádí po malých krůčcích. Zvláště u nezkušených programátorů. Protože skutečně zasahujeme do funkčního kódu a to tak, že nám můžou vzniknout dost záludné chyby. Proto se neustále testuje. Což v XP není problém, neboť každý programátor má vybudovanou sadu testů. Refaktorování se používá v iterativním systému vývoje. XP programátor je až tak daleko, že se každou minutou musí rozhodnout mezi tím jestli refektroruje nebo programuje nový kus kódu. Jako příklad lze uvést; výměna bloku kódu za výkonnější. To, že se systém XP neustále mění, klade obrovské nároky na refaktorovaní. Poněvadž refaktorování systému udržuje stále strukturu přehlednou, funkční a v celkově dobré kondici bez záhadných a nepřehledných tříd a metod, bez nedostupných bloků kódu atd. Bez refaktorování se systém stane křehkým a může se rozpadnout jako domeček z karet. Čím větší projekt, na kterém pracuje mnoho pracovníků, tím je tato náchylnost větší. Oprava systému, která je zaznamenána v tak zbídačeném stavu je pak velice nákladná. Ovšem pokud programátor má perfektně nastudovány principy refaktorizace, potom to není spásou. Jelikož programátor musí být schopen najít takový kus kódu, který je pro refktorovnání vhodný. A na to je nutné mít dostatek praxe.

Jelikož refaktorovnání se již používá několik let, je pro snadnější refaktorování na trhu spoustu utilit a do běžných vývojových prostředí jsou tyto mechanismy zabudovány.

6.5.3. Programování řízenými testy

Spočívá v přístupu vývoji software, který je založen na malých neustále se opakujících krocích, které nás pomalu a jistě posouvají vpřed. Tento postup, i když je na první pohled zbrzděním celého vývoje naopak zefektivňuje celý proces vyvíjení software.

Prvním úkolem je tedy napsat si funkcionalitu a pak test, který funkcionalitu ověří. Až poté se sestaví samotný kód, který se pomocí testu ověří. Těmto testům se říká United testy a jsou to

testy, jenž testuje nejmenší testovatelnou jednotku. Většinou třída nebo metoda. Takové testy se pak píší tak, aby byly schopné automatického testování. Díky tomu jsme budoucnu při provedení další iterace schopni otestovat kompletně celý napsaný projekt. Psaní automatických testů však není triviální záležitosti a je k němu zapotřebí hlubšího zamyšlení.

Postup tedy je:

1. **Napsat test –** prvním krokem, než začneme psát nový kód je, že si sepíšeme požadovanou funkcionalitu například třídy - jaké budou vstupy a co bude vracet. Tím si vývojář naprosto ujasní, jestli chápe, co požadovaný kus kódu má dělat a jaké budou vnitřní postupy. Funkcionalitu a následný test v XP sepíše podle user stories.
2. **Spuštění testů –** po napsání samotných testů je spustíme. A ujistíme se, že žádný z nich neprojde. Pokud by prošel, znamenalo by to, že tuto funkcionalitu již někdo před daným programátorem napsal. A již tedy není třeba dále pokračovat, protože tato funkce je v systému implementována. Tím zamezíme, aby 2 vývojáři dělali jednu a tu samou práci.
3. **Psaní vlastního kódu –** pokud tedy žádný test z bodu 2 neprošel, můžeme začít se samotným psaním kódu. Kód píšeme tak, aby testy prošly.
4. **Test -** v tomto kroku testujeme za pomoci testů z bodu 1. Pokud kód splní všechny testy, je v pořádku a můžeme přejít do dalšího kroku. V opačném případě se vracíme do bodu 3.
5. **Refaktorace –** provedeme refaktoraci tak, jak je popsána v předchozím kroku. Aby pokud možno kód byl „pěkný“, elegantní a přehledný, často využíváme automatizované systémy.

Výhodou tedy je, že v budoucnu, pokud přidáme nějakou komponentu či nějakou funkcionalitu, která nečekaně naruší funkci úplně někde jinde, snadno na to přijdeme pomocí série těchto automatických testů. Pokud chceme změnit někde funkcionalitu, třeba na základě změněných požadavků ze strany klienta, změníme testy a poté měníme až samotný krok.

6.5.4. Týmové praktiky

Párové programování

Programátoři pracují v párech. To znamená, že u jednoho počítače sedí a mají jednu klávesnici a myš. Jeden při vývoji programuje a druhý kontroluje jeho práci. Uvažuje nad

souvislostmi, zda právě psaný kód nemůže ovlivnit jinou část systému nebo již v systému existuje. Dvojice programátorů je různé úrovně znalostí z důvodu, aby se od sebe navzájem učili a tím tak sdíleli informace. Ideální je, pokud se pak jednotliví programátoři mění i ve dvojících; úroveň všech programátorů se značně zvýší. Protože se vlastně časem učí všichni od všech. Další výhodou je, že výsledný kód je částečně zrevidován druhým programátorem, takže dochází k menší chybovosti.

Společné sdílení zdrojového kódu

Sdílení zdrojového kódu v praxi znamená, že žádný z programátorů nemůže nikoho osočit, že udělal chybnou část a nutit ho k opravě. Všichni můžou upravovat všechno. A díky testům v případě chyby dojde k okamžitému odhalení. Tato praktika zapříčiňuje další pružnost týmu, protože nespoléháme na žádného konkrétního člena. Každý z týmu zná prakticky celý vyvíjený systém do detailů. Pokud se objeví nějaká skrytá chyba nebo zádrhel, na který nepřijdou ani testy, můžou ji odhalit samotní programátoři.

Standardizace zdrojového kódu

Každý tým by na začátku projektu měl stanovit jisté standarty, jak se bude psát vyvíjený kód. Laicky řečeno, jaká bude struktura souborů, jak se budou pojmenovávat soubory, třídy, funkce, metody atd. Ve výsledku by tedy nemělo jít ze stylu psaní kódu poznat, kdo jej napsal. Cely kód bude unifikovaný a bude zajištěn i dobrý základ pro provádění refaktorizace.

Tempo vývoje

Rychlost vývoje by měla být v udržitelném tempu. Na tým nelze klást termíny, které není schopen splnit v klasické směně. Přesčasy se za normálních okolností nařizují jen v termínech těsně před vydáním finální verze. Pokud tým neustále pracuje v časovém vypjetí s nedostatkem odpočinku a volného času, následuje přepracování a pracovníci nepodávají tak kvalitní výkony a dělají chyby. Díky nutnosti, co nejvíce v co možná nejkratším čase, si dostatečně nerozmýšlejí, jak která část systému bude fungovat a tak můžou vzniknout dosti závažné chyby.

Pokud skutečně dochází k neschopnosti týmu plnit dohodnuté termíny, je důležité se podívat zpět na plánování, zda dohodnuté termíny jsou reálné. Programátor pracující pod zdravou mírou stresu dělá méně chyb, vytváří kvalitnější produkty a nakonec je celý výsledný výrobek i levnější, protože se sníží cena za poslední část testů a jejich následnou opravu.

6.5.5. Business praktiky

Zákazník na pracovišti

Na pracovišti by měl být dostupný zákazník, nejlépe budoucí klíčový zaměstnanec, který s vyvíjeným sofware bude aktivně pracovat. Zákazník na pracovišti zlepšuje souznění týmů programátorů a zákazníka. Zlepšuje se komunikace a lidé lépe pochopí, co zákazník požaduje. Zákazník určuje priority toho, co se bude v následné iteraci provádět. A určuje další požadavky na vyvíjený produkt.

Plánování hry

Každý projekt se plánuje tak, aby v co možná nejkratším čase vyrobil co nejvíce peněz. Bohužel žádný plán se nesplňuje na 100% a pro dosažení optimálního plánu je nutné návrh neustále upravovat nastaveným podmínkám. Každý projekt má několik releas, které jsou od sebe několik měsíců vzdáleny. Každý releas má nemnoho iterací. Iterace trvá několik dní. Každá iterace má několik úkolů, Úkol většinou trvá v rámci minut maximálně dnů.

Existují tedy 2 základní plány.

Releas plán –

1. Zákazník vytvoří a předá user stories
2. Nacenění jednotlivých user stories
3. Zákazník si vybírá, které user stories chce v nadcházející releas naprogramovat a vybírá tak, aby práce nebylo více než v minulém releas plánu.

Plán iterace –

1. Opět zákazník vytvoří seznam user stories, která chce nechat naprogramovat. Jsou to ty, která byla vybrána do releas plánu.
2. Programátoři z každého user stories vytvoří sadu úkol, které je nutno splnit.
3. Programátoři si vybírají a rozdělují jednotlivé úkoly.
4. Odhaduje se potřebný čas na práci. Čas by měl být plus mínus stejný jak na minulou iteraci.
5. Pokud je práce odhadnuta na příliš dlouhou dobu, zákazník některá user stories z dané iterace vypouští. Nebo naopak přidává.

Vydávání prereleas verzí

XP doporučuje co možná nejčastěji vydávat nové verze. Ty se pak předvádějí zákazníkovi. Díky tomu máme neustále zpětnou vazbu od klienta. Poslední verze je díky neustálému udržování ve funkční verzi připravena k nasazení.

Slovník

Na začátku projektu se definuje slovník. Je to popis vlastností, hodnot, funkcí atd. běžnou každodenní řečí tak, aby v případě, kdy zákazník nebo programátor zmiňuje nějakou část systému, aby oba dva chápali jako stejnou věc se stejnou funkčností. Například frontend – veřejně přístupná část e-shopu. Tak jak vývoj postupuje iteracemi tak se i slovník stále zvětšuje a nabaluje na sebe nové výrazy.

6.6. Proces v XP

V XP není jednoznačně určen proces. Je pouze naznačeno, jak jednotlivé aktivity mají jít postupně za sebou. V XP samotní aktéři vycítí, co by mělo následovat. Pokud ovšem nutně potřebují mít členové přesnou specifikaci procesu, doporučuje XP obrátit se raději na OpenUp.

Ve své podstatě, jak již bylo mnohokrát zmíněno, je XP založeno na velké komunikaci se zákazníkem, častých iteracích a zpětné vazbě od zákazníka na základě právě dokončené iterace. Dále se držíme s Busnisse praktik plánování hry tak, aby projekt byl pro nás rentabilní.

Hrubý nástin procesu v XP jak by měl vypadat je následující:

1. Sepsání vize zákazníka. Vize obsahuje hlavní myšlenky nápady a představy o budoucím software
2. Schůzka se zákazníkem na které se rozepisuje vize do jednotlivých user stories.
3. Zákazník vybere některé scénáře na zpracování v dané iteraci. Programátoři odhadnou čas potřebný na jejich zpracování. A rozhodnou, zda je počet optimální na zpracování v jedné iteraci.
4. Programátoři si jednotlivé user stories rozepíši na úkoly a následně si je rozeberou
   V knize „Extreme Programming Explained“ je neformální popis fází projektu. Aby bylo zřejmé, jak se kdy máme jak zachovat

6.6.1. Zkoumání

Toto je předprodukční fáze, která nepřináší žádné hodnoty. Tudíž je zapotřebí ji zpracovat co nejdříve. V této fázi sbíráme co nejvíce informací a materiálů tak, aby mohly být vytvořeny story cards. Tato fáze končí ve chvíli, kdy již nejsme schopni sehnat žádné další informace a programátoři nejsou schopni odhadnout další fungování systému, aniž by nezapočali práci. Po skončení této fáze se již uvažuje o použité architektuře.

Programátoři nejčastěji volí mezi 3 až 4 možnými architekturami. V týmu si rozdělí zvolené architektury a každý věnuje týden jedné z nich. Na konci každý programátor presentuje zvolený modul a vysvětluje možnosti budování softwaru na právě této architektuře. Programátoři nakonec vyberou favorita a na vybraných podkladech postaví produkt.

V době, kdy programátoři volí architekturu, si zákazník píše jednotlivá user stories. První user stories bude asi pro budování nepoužitelné. Takže je nutné dát co možná nejrychleji zákazníkovi zpětnou vazbu, jak by user stories mělo vypadat.

Výsledná délka předprodukční fáze je od několika týdnů v případě, že se používají již známe technologie, až po několik měsíců u nových technologii.

6.6.2. Plánování

Ve fázi plánování odhaduje termín, kdy bude hotov první releas na základě vybraných user stories zákazníkem. První iterace a následně i první releas je hotov přibližně za 5 až 6 měsíců, tak aby se stihly provést alespoň základní busnisses problémy.

Iterace k prvnímu releas

User stories pro první iterace se vybírají způsobem, aby vytvořily celou kostru systému. Jednotlivé iterace trvají od jednoho do čtyř dnů. Na konci každé iterace máme vytvořenou sadu test case. Na konci prvního releas je projekt připraven na samotnou výrobu, protože je již hotova kostra.

Produkce

Kent Beck k této fázi říka „make it run, make it right, make it fast“. Iterace trvají okolo jednoho týdne. Na začátku každého dne jsou pořádány velmi krátké schůzky, aby bylo ujasněno, kdo se čemu věnuje.

Údržba

Údržba je klasický běžný stav. V této době se experimentuje s výkonem, provádí se refaktorizace. Mohou se přidávat nové funkce do systému. Případní pracovníci, kteří odešli, jsou nahrazování apod.

Ukončení projektu

Pokud zákazník nepřichází se žádným vylepšením, je projekt ukončen.

6.6.3. Příklady použití jednotlivých metodik

Níže je popis jak by se mělo postupovat ve vývoji software za použiti jednotlivých metodik. Veškeré zde uvedené konkrétní informace jsou smyšlené a složí pouze jako ukázka.

Slovník

Zákazník – většinou se jedná o firmu, která poptává zhotovení softwarového díla. Za firmu je většinou jmenována zodpovědná osoba, nebo primární uživatel, který se stará o vyvíjení celého softwarového díla tak, aby splňovalo dané požadavky. Smlouvu o dílo podepisuje majitel, ředitel nebo osoba oprávněná podpisem.

Zhotovitel – je nejčastěji firma, která se smluvně zavázala zhotovením softwarového díla. Zhotovitel je ten, pod kterým pracuje celý vývojový tým. Zhotovitel zaměstnává 3 programátory, 1 testera, 1 externího grafika, konzultanta a projektového manažera.

Smlouva o softwarovém díle obsahuje popis funkčnosti systému a všech doplňkových modulů. Vyvíjený produkt bude v našem případě internetový obchod.

- Internetovový obchod
    - Fungující na 2 doménách. Na každé doméně bude použit jiný jazyk a měna
    - Každá doména bude mít jiný design
    - Internetový obchod bude obsahovat tyto základní funkce (seznam zboží, detail zboží, nákupní košík, objednávka, registrace, multiměnovost, multijazyčnost, seznam kategorií, emailový zpravodaj, nejprodávanější zboží)
    - Administrační část obchodu bude obsahovat (správu administrátoru, vyřízení objednávek nezávisle na doméně, přidání a editaci zboží, přidání a editaci kategorii, cenotvorbu, správu doprav a plateb, import zboží z XML)
- Redakční systém
    - Veřejně přístupná část (kategorie článku, seznam článku, detail článku, reklamní systém, fórum k článku)

- Administrační část (psaní a editace článku, přiřazení článku do kategorie, tvorba a editace kategorií, reklamní systém, editace komentářů ve fóru)

Zákazník již má internetový obchod, ale s jiným sortimentem. Naše firma vyhrála ve výběrovém řízení na zřízení nového e-shopu. Nový internetový obchod bude naplněn dávkovým importem pomocí souboru xml od externího dodavatele zboží. Tento dodavatel doplní také xml soubor, ve kterém bude veškerý prodávaný sortiment včetně stromu kategorií.

Níže tedy budou popsány 3 způsoby realizace a to pomocí SCRUM, OpenUp a extrémního programování. Ve všech případech se zákazník musí rozhodnout, co je pro něj prioritou, zda-li samotný redakční systém nebo e-shop nebo jestli bude společnost dělat práci souběžně, jak na redakčním systému, tak na e-shopu.

7. Postup podle OpenUp

Proces OpenUp je natolik detailně popsán, že držet se jej by nemělo činit nejmenší problém.

7.1. Rozdělení rolí

Analytik:

- Je role na projektu, kterou zastává strana zákazníka a to tím, že získává jeho požadavky a následně je spravuje. Analytik také vytváří slovník, use case a určuje priority.
- Je to člověk, který je dostatečně zodpovědný, komunikativní a vyzná se v dané problematice. Je vhodné, aby tato role povstala z řad programátorů s tím, že musí splňovat výše jmenované vlastnosti.

Architekt:

- Architekt nejčastěji vzejde z programátora. Měl by mít velkou praxi a dostatečně kritický pohled na věc. Avšak pro vybranou architekturu musí umět i motivovat.
- V malých společnostech je role analytika a architekta přiřazena jedné a ta samé osobě. Naopak ve velkých společnostech roli architekta mohou zastávat i dva programátoři.
- Architekt je zodpovědný za výběr nejvhodnější architektury. Anebo její samostatný vývoj.

Programátor:

- Programátor je zodpovědný za vyvíjení části software, která bude později začleněna do finální verze systému. Dále musí být schopen takto začlenit i nové komponenty.
- Po programátorovi se požaduje zkušenost v budování systému v daném programovacím jazyce, schopnost pracovat na zvolené architektuře a technické předpoklady, aby vytvářel složitá technická řešení. Programátor může být sebeschopnější v technických řešeních, ale pokud není schopen komunikace tak jeho práce nikdy nebude dosahovat požadovaných kvalit.
- Programátoři, kteří nedosahují požadovaných kvalit, dělají nejčastěji testery

Projektový manažer:

- Projektový manažer je člověk, který má dostatečnou praxi v budování software. Musí umět udržet týmového ducha. A v případě nouze by měl být schopen vyřešit konflikty.
- Projektový manažer řídi celý projekt ve všech fázích a iteracích těchto fází.

Stakeholder:

- Je to role, která obsahuje několik lidí. Například zákazníka. Jeho úkol je vesměs pasivní, aby byly uspokojeny nastavené požadavky. K uspokojení svých požadavků řídí a plánuje jednotlivé iterace. Podílí se na plánování celého projektu a tvorbě testovacích případů.

Tester:

- Tester je nejčastěji nezkušený programátor, nebo osoba pro tuto činnost speciálně vyškolena. Tester musí zvládat psaní test case a pak je následně spustit a tím testovat produkt. Požaduje se znalost testování software, aby napsal přehledný a zcela jasný report a následně zkontroloval jeho opravení.

7.2. Počáteční fáze

Je to první fáze, která je definována v OpenUp. V počáteční fázi se iteračně sepíše vize, slovník a plán projektu. Tyto dokumenty sepisuje analytik, architekt a projektový manažer a samozřejmě se na nich podílí i zákazník. Vlastnosti softwaru jsou sepsány běžným jazykem od zákazníka. V této fázi se snažíme pochopit problém vyvíjeného softwaru hlouběji.

Slovník – vytvoří jej analytik. Slouží k tomu, aby nenastaly nejasnosti v použitém názvosloví mezi vývojovým týmem a zákazníkem. Například výraz frontend. Zákazník si představí část

e-shopu, kterou vidí neregistrovaný zákazník, ale ve skutečnosti jde o celou část e-shopu, která je přístupná zákazníkům.

Vize – je to dokument, jejž sepsal zákazník a je to jeho pohled na fungování, vzhled e-shopu. Například: V hlavičce e-shopu bude vsup pro hledaný výraz a vedle něj tlačítko, po jehož stisku bude hledaný výraz vyhledán.

Plán projektu – tento dokument sepisuje projektový manažer. V plánu projektu jsou definovány jednotlivé iterace, které budou na projektu provedeny, který z členů zastane jakou roli, jak dlouho jednotlivé iterace budou trvat atd.

V této fázi projektu jsou velice důležité osobní schůzky mezi zákazníkem, analytikem, architektem a projektovým konzultantem. Na těchto schůzkách probíhá upřesňování jednotlivých funkcí e-shopu. Například jak budou rozloženy jednotlivé panely, kolik informací je nutno zapisovat na výpise zboží aj. Po skončení této fáze bude mít zákazník představu o finální ceně projektu. Strana, která se stará o zhotovení softwaru má ujasněné požadavky na e-shop včetně funkčnosti. Projektový manažer celý tuto fázi sleduje aby mohl v příštích fázích blíže určovat postup prací. Zda bude vhodnější začít tvorbou databáze nebo zvolením architektury nebo úplně jiným krokem.

7.3. Upřesňující fáze

V této fázi implementace eshopu si jednotliví členové týmu prohlubují znalosti o vyvíjeném softwaru. Architekt v této části navrhuje vhodnou architekturu a programátoři na ní začnou pracovat. Ty části, které programátoři dokončí, jsou v této fázi již testovány. Upřesňující fáze je velice důležitá. Protože je nutno na jejím konci zkontrolovat, zda zvolená architektura odpovídá požadavkům zákazníka. Prací analytika v této fázi je, aby rozšířil specifikaci, slovník a staral se o rozpis jednotlivých use case. Tester v této fázi kromě testování architektury píše nové test case na základě use case od analytika.

Pokud to vezmeme konkrétně, co bude na e-shopu v tuto chvíli hotovo. Jde o přidání a editaci zboží a struktura databáze. Veškeré funkční prvky, které jsou zobrazeny, jsou ve své nestylované podobě, pouze tak, jak je vložili programátoři.

7.4. Konstrukční fáze

V této fázi je analytikovi většina funkcí zcela jasná. Probíhá upřesňování nejmenších detailů, nebo zákazník dává požadavky na nové funkce. Například informaci, který z administrátorů vyřizuje danou objednávku.

Programátoři „jedou“ podle technické specifikace. Naprogramovanou část (například registrace zákazníka) následně testují. Po zhotovení části software a jejím otestování (v této fázi testuje programátor sám po sobě), je tato funkce přidána do software. To, co je pak následně integrováno do software, je testováno testery. Ti si vytvářejí testovací skripty, které slouží k opakovanému testování části software.

Konstrukční fáze končí tím, že všechny požadavky, jež jsou kladeny na software ze strany zákazníka, jsou zpracovány a následně otestovány.

7.5. Předávací fáze

V této fázi je produkt již připraven k ostrému nasazení. Je předváděn klientovi a ten navrhuje drobné úpravy, které jsou zpracovávány. V této fázi se mohou ladit také ještě drobnosti, co se například výkonu týče nebo komfortu uživatelského prostředí. (jednoznačnost hlášení, přehlednost atd.)

8. Postup vývoje dle SCRUM

Scrum se víceméně zabývá pouze manažerskou části vývoje software.

8.1. Role podle SCRUM

- **Product owner –** Je to osoba obeznámená s projektem a který je na pomezí analytika, konzultanta a projektového manažera. V našem konkrétním případě budeme volit konzultanta. Product owner musí být často v komunikaci se zákazníkem a je vhodné, aby věděl, v jakém oboru podniká klient. Mezi jak velké patří a jaké jsou jeho požadavky a specifika oboru. To vše na základě pochopení požadavku a zlepšení jeho provedení.

- **Scrum master –** je jinými slovy vedoucí týmu. Stará se o každodenní běh týmu. V praxi to znamená, aby měl tým zajištěno vše pro své každodenní fungování. Řeší neshody v týmu, řídí projekt, odstraňuje nastalé překážky.
- **Scrum tým –** je to tým lidí z různých profesí, kterým jde primárně o vývoj nového produktu. Patří zde například programátor, databázista, tester, kodér, grafik ale také product owner a scrum master.
- **Stakeholder -** je zákazník, nebo klíčový uživatel. Je nutné aby byl dostupný na pracovišti, kde probíhá vývoj produktu. Určuje funkcionalitu a postup práce a hájí zájmy své strany.

8.2. Proces

První fáze je taková, že se sejde zákazník s product ownerem a sepíší spolu požadavky na budoucí software. Žádosti se sepíší klasickým jazykem, aby byl všem stranám srozumitelný. Tento seznam požadavků pak přechází na product ownera. Ten jednotlivé požadavky projde a na základě dokumentu vytvoří sprint backlog dokument. Sprint Backlog dokument je formální sepsání zákazníkových požadavků. Tuto listinu si pak projde zákazník bod po bodu a stanoví priority zpracování. Například je pro zákazníka prioritou možnost správy zboží, editace kategorií a dávkový import tak, aby si mohl připravit databázi zboží. Až se určí priority, poté jsou požadavky zařazeny do jednotlivých iterací. Na každou iteraci je určen daný čas. A úkoly by měly být v navrženém čase zvládnuty.

	Název	náročnost
Vysoká priorita		
1	vytvoření databáze eshopu	30
2	tvorba a editace zboží	25
3	dávkov import	50
střední priorita		
4	návrh a implementace uživatelkého prostředí	100
5	funkce pro funkční eshop	120
nízká priorita		
6	CMS systém	80
7	statistiky	20

Obrázek 10: Product backlog

Každý sprint tedy začne tím, že zákazník určí, které úkoly v něm budou provedeny. Tudíž klient pouze vybere z product backlogu úkoly a přesune je do sprint backglogu. V našem případě počítáme s tím, že během jednoho sprintu trvající týden se stihne 105 jednotek práce.

	Den sprintu / Zbývající práce				
Úkol	1	2	3	4	5
	105	85	68	53	38
Vytvoření databáze eshopu					
zjištění informací od zákazníka	5				
Návrh db (ERD)		15			
Vytvoření db			10		
Tvorba a editace zboží					
Diskuse se zákazníkem		2			
návrh funkčnosti			5		
implementace				15	
testování					3
Dávkový import					
Diskuse se zákazníkem	5				
Kontaktování dodavatele	5				
návrh funkčnosti	5				
implementace		10	10	10	
testování					5

Obrázek 11:Sprint backlog

V tabulce sprint backlogu vidíme, že veškerá naplánovaná práce by se měla stihnout i s rezervou. Pokud by se práce nestihla, znamená to, že odhad množství práce na úkolu je špatný, nebo tým zvládne za 5 dní menší počet jednotek práce. Pokud by skutečně taková situace nastala, řešila by se právě na sprint review. Uvedený meeting slouží k zhodnocení právě ukončeného sprintu.

9. Postup podle extremního programování

XP se přímo nezabývá a neřeší komplexní proces vývoje, ale namísto toho se zabývá technikami samotného vývoje.

9.1. Role

Dle XP musíme definovat minimálně tyto role.

XP coach – je podpůrná role. Tudíž osoba obsazena na tento post může vykonávat více rolí. Hlavním úkolem XP coache je kontrola dodržování metodik.

XP zákazník – je to samotný zákazník, nebo primární uživatel budovaného e-shopu. Hlavním požadavkem je, aby měl představu o tom, co vlastně chce.

XP programátor – členové vývojového týmu

XP tester – tester, jeho funkcí je to samé jako u SCRUM a OpenUp

XP tracker – Má za úkol sledovat pokroky ve vývoji software.

Jak již bylo zmíněno dříve v popisu XP základem u XP je, že se programuje v párech. Proto je vhodné mít sudý počet programátorů. Členové v párech se obměňují tak, aby se šířily znalosti mezi všechny členy. Dále je vhodné, aby nezkušení programátoři tvořili pár se zkušenějšími členy týmu.

9.2. Průběh procesu

Proces probíhá v krátkých krocích tak, aby byla co nejčastější zpětná vazba od zákazníka. Více definic procesu v XP není zapotřebí.

9.3. Fáze

9.3.1. Zkoumání

Karta uživatelského příběhu	
Datum: **10. 7. 2011**	Oprava: ☐ Nový: ☐ Rozšíření: ☐

Číslo příběhu: 23	Priorita:
Riziko:	Odhad náročnosti:
Popis: Pokud uvidím v administraci přijatou objednávku. Která je zpracovávána nebo již vyřízena. Potřebuji vidět, kdo ze zaměstnanců tuto objednávku vyřizuje. Důvod je prozaický. Nikdo se pak nemůže vymlouvat, že tuto objednávku nevyřizuje, nebo o ní neslyšel.	
Poznámky: 	

Obrázek 12:Uživatelský příběh

Datum	Stav	Udělat	Komentář

V této fázi je nejdůležitější úkol na straně zákazníka. Ten musí vytvořit jednotlivé uživatelské příběhy. Uživatelské příběhy popisují fungování software. Pokud zákazník tvoří uživatelské příběhy poprvé v životě. Je mu nutno ze začátku pomoci s tvorbou. Níže je ukázka, jak takový uživatelský příběh vypadá. V tuto chvíli, kdy zákazník píše uživatelské příběhy, testují programátoři budoucí softwarové zázemí. Kupříkladu u e-shopu se testují možnosti použité technologie webového serveru, na které e-shop bude spuštěn, pro jaké prohlížeče bude e-shop optimalizován a zda-li je nutné pro stejný počet prohlížečů optimalizovat i administraci.

9.3.2. Plánování

V této fázi se plánuje, za jak dlouho bude vydána první funkční verze softwaru. XP doporučuje dobu 2 až 6 měsíců. Jelikož v našem případě jde o nepříliš náročný e-shop, první iterace bude uveřejněna přibližně za 3 měsíce.

9.3.3. První iterace

Zákazník určí, které příběhy jsou pro něj nejdůležitější. Znamená to, že tým je obeznámen s daným seznamem důležitých příběhů, který si rozdělí. Jednotlivé iterace podle XP by měly trvat okolo 4 týdnů. Jelikož v našem případě máme hodně menších příběhů, bude vhodnější použit iteraci okolo 2 týdnů. Po skončení první iterace bude mít tým již zvolenou architekturu. Jak již bylo napsáno výše, před samotným programováním se nejprve píšou testy.

9.3.4. Produkce

V této fázi se jednotlivé iterace zkracují až na délku jednoho týdne. Jinak je práce beze změny, pouze priorita uživatelský příběhů je nižší.

Po dokončení projektu probíhá údržba; fáze ladění detailů, zvyšování výkonu a zdokonalení bezpečnosti. Případně se provádějí drobné úpravy na popud zákazníka. Tato fáze končí ve chvíli kdy zákazník, již nepíše žádné další připomínky. Systém se pak pokládá za ukončený.

10. Rozdíly jednotlivých metodik

10.1. Rozsáhlost metodik

Nejvíce rozsáhlou metodikou je OpenUp. Jejích rozsah a komplexnost se nedá vůbec srovnat se Scrum a xtreme programing, které obsahují jen několik artefaktů a rolí.

Bohužel však rozsah může být občas i odstrašující. Ale na druhou stranu opravdu jen výjimečně nastane situace, kdy v procesu nevíte jak dále pokračovat.

OpenUp, jako jediná, není čistě agilní metodika. Někteří ji spíše řadí k tradičním metodikám.

10.2. Popis procesu

OpenUp má naprosto dokonale specifikován každý proces. Procesy mají své fáze. Oto popsání procesu také řadí OpenUp k tradiční metodikám.

Scrum nemá přesně popsán proces. Obsahuje jen diagram pode, kterého se máme řídit. Podle tohoto diagramu se můžeme docela dobře řídit. Proces se dělí do jednotlivých kroků, které se iterativně opakují tak, až se přiblížíme zdárnému konci.

XP má proces naznačen velice chabě. Pouze jako iterativní kostra. Důvod je jednoduchý. XP je vlastně soubor nejlepších zkušeností. Proto je XP vhodné kombinovat s dalšími metodikami, které mají proces lépe popsaný.

10.3. Souhrnné informace

10.3.1. OpenUP

Je řazeno na pomezí agilních a tradičních metodik, i přesto se více přiklání k tradičním. Jelikož fáze procesu jsou jasně odděleny a pokud přejdeme, z jedné fáze do další, už se nevracíme zpět. Oproti SCRUM a XP je tato metodika dosti robustní. Převážně tím, že je popsán do detailů. Například popis rolí, artefaktů, procesů, aktivit se nedá srovnávat svým rozsahem se Scrum a XP.

To, že se v OpenUp v jednotlivých fázích již nevracíme zpět je důvod, proč je nutné věnovat obrovské úsilí přípravě. Je definována role architekta. Ta nese obrovskou tíhu zodpovědnosti. Protože špatně zvolená architektura může v budoucnosti přinést značné problémy.

Testy v OpenUp probíhají až nakonec. Nakonec je důležité říci, že OpenUp se dá kombinovat s XP.

10.3.1.1. Klady

Hlavní výhodou oproti tradičním metodikám je iterativní přístup. Iterativní přístup v OpenUp znamená, že je vývoj produktu rozdělen na několika menších fázích, které se provádějí v iteracích. Na konci každé iterace je produkt předveden zákazníkovi. To zajišťuje, že výsledný produkt je opravdu tím, který si zákazník objednal a představoval.

V případě tradičních metodik, není možné jít zpět. A jelikož neobsahují ani iterace, hrozí, že pokud se ve fázi zjišťování požadavků a v následném vývoji zákazník nepochopí s vývojovým týmem, stane, neexistuje možnost opravy.

To, že v OpeUp obsahuje iterace také znamená, že je již v raných fázích projekt funkční. Klient si jej může osahat, což značí, že dílo má tzv. přidanou hodnotu.

10.3.1.2. Zápory

I když jsem psal, že OpenUp je velice komplexní a může být složité jej nasadit, oproti RUP je jeho složitost značně zjednodušena. OpenUp je na rozdíl od RUP ochuzena o různé dokumentace, doklady, analýzy atd. Proto v OpenUp negenerujeme tolik dokumentů. Z tohoto vlastně zjednodušeně vyplývá, že OpenUp je možno nasadit i na menší projekty s menším počtem lidí.

Hlavním nevýhodou OpenUp se tedy stává jeho přílišná složitost. Což při zavádění v již fungující společnosti může způsobit vlnu nevole v řadách zaměstnanců.

10.3.2. Scrum

Scrum se opět velmi často kombinuje s XP. Důvod je jednoduchý. Scrum se nezabývá již přesně samotným vývojem software. Jde v něm jen o jednoduchý popis vývoje software. Je jen nastíněno, že vývoj by měl být v iteracích a že bychom měli být neustále připraveni na možné změny, které přijdou ze strany zákazníka.

10.3.2.1. Klady

Hlavním kladem Scrumu je jednoduchost. Což značí jeho nasazení do již zaběhlé společnosti za „bezbolestné“. Scrum neobsahuje příliš rolí ani artefaktů, převážně díky multifunkčním rolím. Často díky stmelování týmu. Protože jednotliví členové jsou neustále v kontaktu, což je dost velký rozdíl oproti OpenUp, kde každá role vykonává jen pevně stanovenou práci.

10.3.2.2. Zápory

Scrum se prakticky nezabývá samotným vývojem. Pokud tedy tým požaduje popis procesu samotného vývoje je dobré, jak již bylo řečeno, Scrum spojit s XP. Scrum se tedy nezabývá fázemi, jak vyvíjet software, jak testovat atd.

10.3.3. Extreme programing

Oproti Scrumu se XP zabývá prakticky jen samotným vývojem. A nikoliv tím, jak dojít k náležitému návrhu na vývoj. XP má na světě hodně kritiků. Hojně kritizovanou oblastí se stává situace, zda je nutné mít dva programátory u jednoho počítače, zda je opravdu nutné nejprve psát testy a jak je vůbec použité XP ekonomicky výhodné. Dále se objevují dohady, že XP je vhodné jen pro senior programátory a je zapotřebí u ní zavést hodně změn atd.

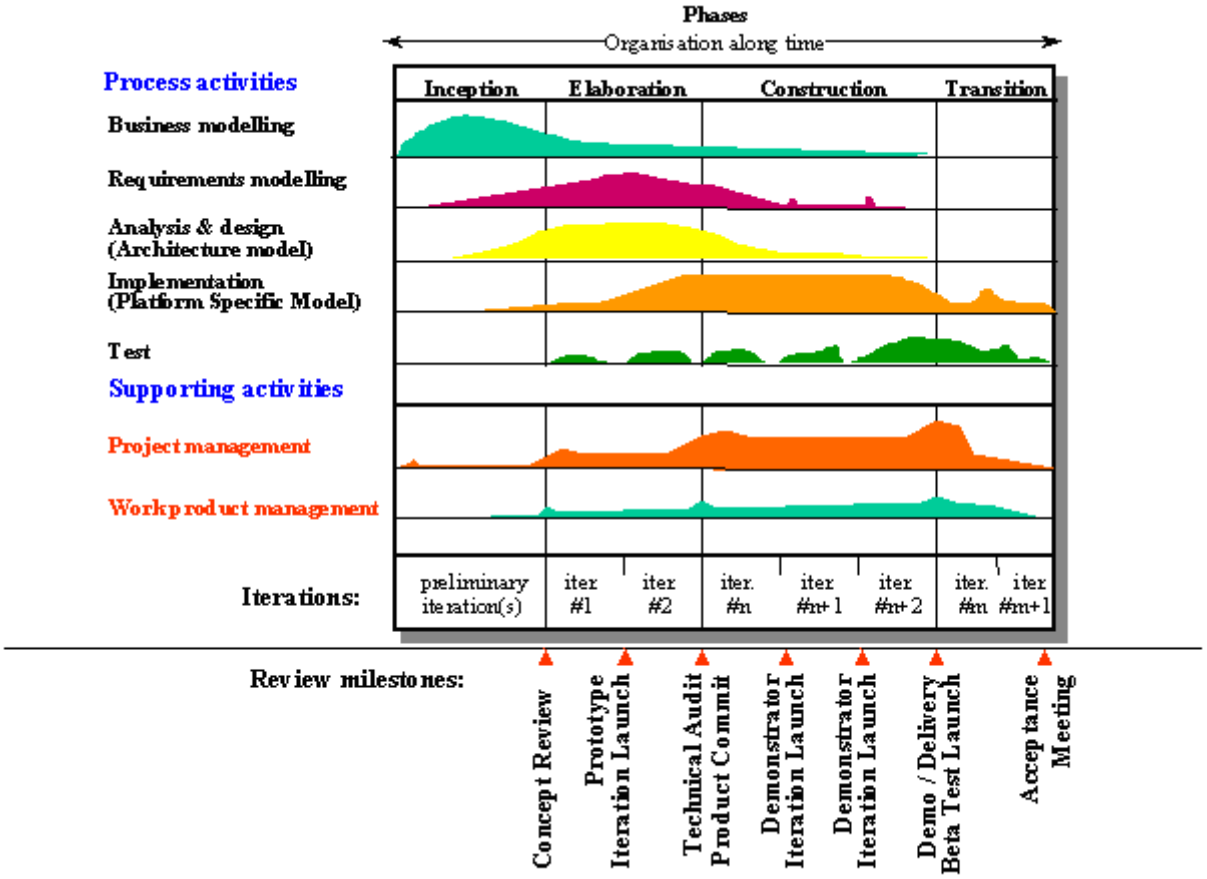
11. Eclipse Process Framework(EPF)

11.1. Základní informace o EPF

EPF vyvíjí skupina okolo eclipse foundation. Jako vše u této skupiny je i EPH opesourcový projekt. EPF slouží k zmapování procesů a jejich modelování.

Výstup z EPF se nejčastěji představuje jako webová stránka, kde následovně jednotliví účastnici naleznou svou roli; jakou mají zodpovědnost, jak mají v případě potřeby dále postupovat, kdo je jejich nadřízený a jaké jsou vazby.

EPF samozřejmě odděluje jednotlivé procesy od sebe. Jednotlivé procesy jsou tedy popsány rolemi, artefakty a aktivitami. EPF pak obsahuje dvě osy. Horizontální a vertikální. Z horizontální vyčteme charakteristiku procesu. A z vertikální jeho obsah. V první fázi je nutné v EPF vytvořit obsah,(role, aktivity). Až je vytvořen tento obsah, poté se vytvářejí jednotlivé procesy.



Obrázek 13: obsah a proces

12. Popis procesů v netdevelo s.r.o.

Společnost netdevelo s.r.o. byla založena v roce 2007. Avšak založení společnosti předcházelo 3leté působení společnosti Shopsys. Jediným majitelem společnosti je pan Petr

Svoboda, který jim zůstal i po převedení společnosti na netdevelo s.r.o. Firma pod sebe převedla samozřejmě i všechny závazky a zaměstnance.

Hlavním oborem firmy je výroba profesionálních internetových aplikací. Ve společnosti se vyrábí jak malá řešení, tak velké individuální B2B portály. Základním produktem je ShopSys, Který je stavěn jako robustní aplikace pro internetový prodej. Obsahuje velké množství modulů s možností individuální změny designu a možností dávkového importu dat od dodavatele. Aplikace ShopSys je i přes svou rozsáhlost vytvářena tak, aby byla snadno upravitelná na míru potřebám zákazníkům.

Ve společnost netdevelo s.r.o. aktuálně pracuje kolem 40 zaměstnanců plus asi 10 externích pracovníků. Většina zaměstnanců dosáhla vysokoškolského vzdělání v oboru IT nebo příbuzných. Ti, kteří ještě vysokoškolského vzdělání nedosáhli, při zaměstnání také studují. V této aktivitě jsou podporování i ze strany zaměstnavatele. Většina zaměstnanců již měla předchozí praxi s vývojem software, provozováním webserveru, či webdesignem. Proto je většina zaměstnanců odborníkem ve své profesi.

Tato diplomová práce se zabývá 4 hlavními procesy. A to výrobou designu, výrobou nového projektu, výrobou nové stabilní verze a fakturačním systémem.

- Výroba designu je proces od získání požadavků na design od zákazníka, které pokračuje kreslením až po implementaci designu na e-shop.
- Výroba nového projektu je proces výroby nového projektu na základě požadavku zákazníka. A nebo výroba případných úprav na již existujícím e-shopu.
- Výroba nové stabilní verze je proces, při němž se vyvíjí nová stabilní verze aplikace ShopSys.
- Fakturační systém je popis procesu získávání peněz za provedené dílo.

12.1. Jednotlivá oddělení společnosti Netdevelo s.r.o.

Společnost se dělí do několika logických oddělení. Každé oddělení má svého vedoucího, který je za jeho běh zodpovědný.

- Výrobní oddělení (programátoři)
- Oddělení designu
- Oddělení napojení
- Oddělení správy serveru
- Konzultanti

- Obchodní oddělení
- Vývoj

12.2. Výroba designu

Tato část práce je neveřejná a je obsažena v neveřejné příloze.

V příloze můžete nalézt popsaný proces výroby designu a jeho následné nasazení, včetně artefaktů a rolí. Celý popis vychází z praktické části

12.3. Tvorba a úprava nových a stávajících projektů

Tato část práce je neveřejná a je obsažena v neveřejné příloze.

V příloze můžete nalézt popsaný proces vývoje nového projektu a nebo tvorby nové úpravy stávajícího e-shopu. Jsou popsány role a artefakty. Celý popis vychází z praktické části.

12.4. Výroba nové stabilní verze

Tato část práce je neveřejná a je obsažena v neveřejné příloze.

V příloze můžete nalézt popsaný proces vývoje nové stabilní verze aplikace e-shop. Jsou popsány role a artefakty. Celý popis je vytvořen v návaznosti na praktickou část.

12.5. Fakturační systém

Tato část práce je neveřejná a je obsažena v neveřejné příloze.

V příloze můžete nalézt popsaný proces fakturačního systému. Popis jednotlivých fází, role, artefakty. Celý popis je vytvořen v návaznosti na praktickou část.

13. Shrnutí aktuálního procesu

Tato část práce je neveřejná a je obsažena v neveřejné příloze.

Je to tedy neveřejné shrnutí popsaných procesů. Je pak zkoumáno, jestli se používané procesy ve společnosti dají přirovnat k některé z popisovaných metodik.

14. Návrhy na zlepšení aktuálních procesů

Po provedení zmapování existujících procesů a jejich následnému porovnání jsem se pokusil o provedení změn v procesech. Bohužel nedošlo k úplnému pochopení těchto změn ze strany kmenových zaměstnanců. Proto jsou navrhnuté změny menšího rázu.

14.1. Kritické faktory procesu

Ve společnosti netdevelo s.r.o. neexistuje metrika, dle které by se dalo určit jak moc byl proces úspěšný. Proces totiž není důležité pouze plnit, ale v kritických místech je nutné mít možnost kontroly, zda vše funguje správně. Každá firma by si tedy měla zvolit několik kritických míst, které jsou pro fungování firmy nezbytné. Taková místa musí splňovat několik základních podmínek.

- jsou realizovatelné
- jsou ovlivnitelné
- jsou měřitelné
- vyzývají k aktivitě
- jsou navzájem nezávislé
- jsou v souladu s posláním

V společnosti netdevelo s.r.o. byly stanoveny tyto klíčové faktory procesu jako doba, která trvá obchodníkovi než představí klientovi specifikovanou nabídku, včetně nacenění. Aktuálně je tento proces dosti neefektivní a pomalý.

Dalším kritickým momentem procesu, který je i měřitelný, se stává délka trvání trvá a počet osob, které musí projít tiket technickou podporou, než se dostanou ke kompetentní osobě, která jej vyřídí.

14.2. Týdenní reporty

Každý zaměstnanec jednou týdně svému nadřízenému sepisuje, jaký měl pracovní týden. Jaké pracovní problémy řešil, zda mu chyběly k jeho práci nějaké podklady či informace, jak se mu komunikovalo z ostatními pracovníky a zda nastaly nějaké nešvary.

Z tohoto dokumentu pak vedoucí mohou vyvodit nějaké závěry. Opakující problémy mohou řešit. Například osobním zajištěním materiálu od zákazníka, domluvou v případě problému mezi zaměstnanci apod.

Týdenní report je inspirován metodikou Scrum a jejími reporty. Týdenní report se na pracovišti osvědčil, protože problémy na nejnižší vrstvě firmy takto mnohdy prosáknou až do nejvyšších pater a dostanou se k řešení.

14.3. Knowledgebase

V společnosti se zavádí databáze znalostí. Ta se dělí na dva logické celky. První z nich bude obsahovat veškeré procesy společnosti tak, jak jsou vytvářeny v této práci. Zmíněná část bude obzvlášť vhodná pro nové zaměstnance. Ti v případě, pokud nebudou vědět jak postupovat v další fázi, naleznou, v jaké fázi procesu se nacházejí právě teď a jaké artefakty potřebují pro další postup, nebo na koho dalšího se mají obrátit.

Druhou část znalostní báze bude uživatelsky vytvářená část. Ta se bude dělit na dvě základní fáze.

- Základní úkony
- Uživatelské rady

V základních úkonech bude popis, jak funguje například svn a jak sním pracovat. Nebo jak se používá nová úprava z modulu. Tyto úkony pak dále budou napojeny na školící centrum v aplikaci, které bude sloužit k vzdělávání zaměstnanců tak, aby jejich kompetence v popisu funkčního místa mohly být rozsáhlejší.

Druhá uživatelská část bude obsahovat články „jak na to“. Budou je vytvářet jednotliví zaměstnanci a ti jsou motivováni motivačním systémem, který je za napsání kvalitního článku odmění. Budou zde například články jak na fotny pomocí fotn-face.

Tyto podobné artefakty se nacházejí ve všech metodikách SCRUM, OpenUp a extremním programování.

14.4. Schůzky realizačních oddělení

U některých úkolů, jenž se provádějí v rámci realizace nového projektu, je nutná spoluúčast více oddělení. Jelikož však každé oddělení sedí v jiné kanceláři, nastává problém s komunikací, kdy není zcela jasné, kdo jakou práci udělá. Kodéři občas dělají drobné programátorské úpravy a programátoři na tuto skutečnost spoléhají.

Proto je nutné zavést pravidelné jedno týdenní schůzky vedoucích jednotlivých oddělení, kteří popíší stavy prací na jednotlivých projektech. Tyto informace získají od svých podřízených. Neprobírají se úplné detaily, ale zajistí se například lidé k celým blokům, nebo se dohodne postup práci tak aby nedošlo k případným konfliktům.

Tato schůzka je inspirována schůzkou z metodiky Scrum – Daily scrum.

14.5. Refaktorování a testy

V současné době neexistují ve společnosti automatické testy. Hlavním důvodem je, že jsou úpravy e-shopu velice individuální. Ale i přesto se pokusíme vytvořit sadu automatických testů alespoň na základní třídy. Díky zmíněným testům pak případné budoucí zásahy do těchto tříd nebudou tak nebezpečné.

Díky automatickým testům by pak bylo i samotné refaktorování snazší, protože po refaktorování ihned můžeme provést automatický test.

Tento návrh přichází z XP.

14.6. Standardy v kódu

Jelikož ve společnosti netdevelo v současnosti na jednom projektu pracuje mnoho osob, je zřejmé, že každý má svůj osobitý rukopis. Standardy kódu slouží k tomu, aby všichni zúčastněni ještě před začátkem práci věděli, jaké standardy mají dodržovat. Například, že editoři mají mít nastavený na kódování UTF-8, tabulátor má opravdu dělat tabulátor a ne řadu po sobě jdoucích mezer. Je také nutné, aby se dodržovaly názvy tříd, metod odsazení podmínek a cyklu.

Vytvoření tohoto artefaktu je podmínkou XP.

15. Závěr

V této diplomové práci jsem v první fázi popsal agilní a jednu tradiční metodiku. Shrnul jsem rozdíly mezi jednotlivými metodikami vývoje software a jaké prvky jsou u těchto metodik společné. Výsledkem tohoto porovnání je, že tradiční metodiky lpí na dokumentaci, artefaktech či klasických formálních schůzkách, tak agilní metodiky mají zcela opačný přístup. A co není nezbytně nutné se snaží vyřadit. Vše tedy musí být elegantní, jednoduché a

především efektivní. Jsou zavedeny iterace, na jejichž konci zákazník vždy může vidět funkční produkt. Díky tomu má vývojový tým zpětnou vazbu, která je častokrát vše říkající a nenastává problém, že zákazníkovi je dodán jiný software, než si objednal.

V práci jsem také použil EPF (Eclipse Process Framework), který jsem využil k mapování procesů společnosti netdevelo s.r.o. Tento software umožňuje posléze vymodelované procesy exportovat do html. Taktéž jsem využil Knowledgebase, která se reálně zavedla v společnosti. Díky tomu se dnes nově příchozí zaměstnanec rychleji zorientuje a může podávat kvalitnější výkon. Procesy vytvořené v EPF jsou neveřejnou přílohou této práce.

V samotném procesu jsem také vytvořil několik zlepšení. Leč se samotným nasazením jsem se setkal s menší vlnou nevole, protože mnoho zaměstnanců dnes fungující proces pokládá za zcela vyhovující. Avšak vedení se optimalizace procesu natolik zalíbila, že v budoucnu se budu podílet na vytvoření a následné optimalizaci kompletního procesu. Optimalizace by pak měla přispět k snížení nákladů ve formě efektivnější práce.

Během této práce jsem se podrobně seznámil ze všemi popisovanými metodikami. Veškeré důležité materiály však k nim jsou pouze v angličtině, tudíž jsem byl nucen vycházet z těchto materiálu, načež ke konci psaní této práce jsem zjistil, že například společnost Seznam.cz podporuje Scrum tím, že nabízí přednášky. Samotné modelování procesu v reálné firmě bylo velice zajímavé. Protože člověk si při něm uvědomí veškeré aspekty, které pro průběh kvalitního procesu je zapotřebí. Provést zlepšení na procesu také nebylo úplně jednoduché, protože mnou navržená změna na přechod k Scrumu nebyla příliš kladně přijata. Musel jsem provést několik menších za to velice pozitivně přijatých změn, které jsou dopodrobna popsaný v neveřejné části práce.

16. Seznam použité literatury

1. Wikipedia.org [online], posled. Revise 8.8.2012 <http://cs.wikipedia.org/wiki/Vodopádový_model>
2. Eclipse Process Framework [online] <http://dev.eclipse.org/mhonarc/lists/epf-dev/pdfIMlVaoFxb1.pdf >
3. Zdojak.cz 18.12.2009 [http://zdrojak.root.cz/clanky/agilni-vyvoj-scrum/]
4. Wikipedia [online], [http://cs.wikipedia.org/wiki/Refaktorování]
5. BECK, Kent - Extreme Programming Explained, 1. vydání 29.8.1999. 224 s. ISBN: 0201616416

6. STEPHENS, Matt - Extreme Programming Refactored: The Case Against XP, Apress. ISBN: 1590590961
7. IBM Rational Method Composer: Part 1: Key concepts [online] 2005-12-5, http://www.ibm.com/developerworks/rational/library/dec05/haumer/
8. Eclipse Process Framework [online] http://www.eclipse.org/epf/general/OpenUP.pdf

17. Přílohy

Součástí této práce je také CD, které obsahuje:

- tento text v elektronické podobě v souboru PDF.
- Kopii zadání této práce v PDF.

Pro potřeby hodnocení vedoucího a oponenta této práce a pro potřeby komise, existuje i papírová příloha, která obsahuje neveřejnou část práce. Úloha také obsahuje praktickou část – export procesů z EPF Composer do podoby XML, který je možné zpět do EPF Composer importovat. A dokument obsahující doporučení pro psaní kódu v společnosti netdevelo s.r.o. Praktická část práce je také neveřejná.